TNO Physics and Electronics Laboratory

Oude Waalsdorperweg 63
2597 AK The Hague
P O. Box 96864
2509 JG The Hague
The Netherlands

Fax +31 70 328 09 61
Phone +31 70 326 42 21

TD

(1)

TNO Defence Research

TNO-report

copy no	title
FEL-93-B029	PRACTICAL: Planning and resource allocation in C²-domains with time critical algorithms

PRACTICAL: Planning and resource allocation in C^2-domains with time critical algorithms

author(s)
C.W. d'Huy
A.P. Keene

date
February 1993

classification	
title	ongerubriceerd
abstract	ongerubriceerd
report text	ongerubriceerd
appendices A-B	ongerubriceerd

no of copies	40
no of pages	94 (including 2 appendices, excluding RDP and distribution list)
no of appendices	2

TNO

Management summary

Planning is an equally important as complex problem in the domain of Command and Control. Given an increasing need for fast and flexible solutions for the allocation of resources, scheduling of missions and planning (large-scale) operations the assistance of human operators with automated tools for scheduling and planning becomes mandatory. Scheduling problems concern a known set of activities and resources that have to be fitted in a fixed time window in accordance with a set of constraints restricting the potential solutions. Planning problems typically involve selections of sets of activities and resources that are not known beforehand that have to be ordered relatively in accordance with constraints in a temporal context that may reach far into the future. Moreover, replanning is a central activity to incorporate continuous changes in the environment into the evolving plan. It is furthermore important to be able to assess the implications of options in terms of "what-if" analyses.

During the generation of a plan choices have to be made. For instance when several alternatives exist that all realise the same goal. One can view the evolution of a plan as the plan's history. We state that this history, together with the plan itself and the reason why the plan should be reviewed are the parameters of the plan review process. Therefore an important part of our research concentrates on means of analysing and using this history. It is plausible that the reason that a plan should be reviewed plays an important role in the reviewing and replanning process.

This report describes a framework for (re)planning using a scenario-driven approach that centres on methods and techniques for plan generation, plan repair and replanning with (partial) re-use of the original plan. The basis for the framework is a conceptual model of the (re)planning process and a knowledge partitioning that distinguishes between strategic, tactical, causal and world knowledge. A temporal database management system is used to allow reasoning about the past and present as well as the future. The research has culminated in the system PRACTICAL, demonstrating the (re)planning framework, incorporating the demonstrators CALIGULA and ALLOCATOR for resource allocation and scheduling respectively.

In the Command and Control context a plan is usually developed as an answer to a certain event or series of events taking place in the world. The approach based on scenarios and scripts is both natural as well as flexible, allowing the refinement of a plan via scripts containing goals to be fulfilled, actions to be undertaken to achieve these goals together with the allocation of resources

to the actions. Moreover, it is a "piecemeal" approach to planning in the sense that it allows partial plan development that can be managed by the temporal database. This makes it possible to incorporate insufficient or uncertain data in partial plans that may be expanded at a later stage.

Another point of interest is the use of algorithms to solve combinatorial problems. Since these are strongly related to resource allocation, an efficient method for solving these problems is important in a C^2-planning process. Again, means of keeping track of the solution path (history) are important when these algorithms are incorporated in the planning/replanning process. Static resource allocation is addressed, whereby time plays no role other than a need for a speedy solution, and has resulted in the demonstrator CALIGULA for the problem of allocating frequencies to a radio link network. The problems in the domain of scheduling are dealt with, which has resulted in the demonstrator ALLOCATOR for the allocation of pilots to missions. Also, dynamic resource allocation during (re)planning is discussed, specifically the problem of sharing limited resources by different activities.

An automated system always raises questions concerning the performance bottleneck. PRACTICAL does well on this count. This is due to the fact that we use algorithms with complexities in the order of the number of edges times the number of nodes for a (virtual) tree spanning the problem domain. Therefore, the potential exponential growth when planning in ever larger domains does not imply an equivalent computational growth when solving the planning problem. Memory requirements for the domain and for the context saving mechanism used pose no problems for state-of-the-art computers.

Another potential bottleneck for a knowledge-based system pertains to knowledge acquisition. In particular, the scripts entail a considerable compilation effort necessary prior to being able to benefit from a scenario-driven approach to planning. However, this is true in general because scenarios and scripts describing e.g. red-white-blue force command and control options will necessarily have to be available for manually planned missions and operations as well in the form of tactical doctrine. An additional effort is thus necessary to implement the scripts in the formal notation required in the PRACTICAL framework, incorporating preconditions, resource allocations and temporal database instantiations.

Planning systems such as the one described in this report will potentially benefit from other current research projects, and vice versa. In particular, there are several EUCLID (European Cooperation for the Long term In Defence) projects that will start in the near future: RTP 6.1 (Command and Control workstation, accentuating (re)planning), RTP 6.4 (Combinatorial algorithms for resource allocation problems) and RTP 6.5 (Crew Assistant on-board single seat fixed wing aircraft). We also refer to the Army Tactical Command and Control Information System (ATCCIS) incorporating a planning module currently being developed.

The applicability of scheduling and planning systems to C^2-domains is extensive. In a military environment possibilities are air force mission scheduling and both pre- and inflight mission and tactical planning, operations planning for G3-sections at army brigade/division level, damage control on board naval ships and naval operations planning. In a civilian environment applications are for instance in the area of fire brigade support, contingency planning in general, police operations, Coast Guard operations and chemical processing industries. This broad scale of application implies a necessity of accentuating the generic character of frameworks and tools for scheduling and (re)planning. Current research at FEL-TNO in the context of planning in the areas described above is directed at naval damage control (the extension of the prototype system DAMOCLES to the operational on-board system ANDES), the participation in the ATCCIS project, the further extension of the PRACTICAL framework, building an application for air force mission scheduling for operational use in the Squadron Operations Centre (SQOC) and the extension of such a scheduling tool to a generic level (as in PRACTICAL) to be able to incorporate goals, activities, resources, constraints and temporal management facilities for the complete range of C^2-domains.

Report no.	:	FEL-93-B029
Title	:	PRACTICAL: Planning and resource allocation in C^2-domains with time critical algorithms
Author(s)	:	C.W. d'Huy M.Sc. B.Sc., A.P. Keene M.Sc.
Institute	:	TNO Physics and Electronics Laboratory
Date	:	February 1993
NDRO no.	:	-
No. in pow '93	:	704.2

Abstract (ongerubriceerd)

Planning is an equally important as complex problem in the domain of Command and Control. Given an increasing need for fast and flexible solutions for the allocation of resources, scheduling of missions and planning (large-scale) operations the assistance of human operators with automated tools for scheduling and planning becomes mandatory. Replanning is a central activity, allowing the incorporation of continuous changes in the environment into the evolving plan. It is furthermore important to be able to assess the implications of options in terms of "what-if" analyses. This report describes a framework for (re)planning using a scenario-driven approach that centres on methods and techniques for plan generation, plan repair and replanning with (partial) re-use of the original plan. The basis for the framework is a conceptual model of the (re)planning process and a knowledge partitioning that distinguishes between strategic, tactical, causal and world knowledge. A temporal database management system is used to allow reasoning about the past and present as well as the future. The research has culminated in the system PRACTICAL, demonstrating the (re)planning framework, incorporating the demonstrators CALIGULA and ALLOCATOR for resource allocation and scheduling respectively.

Rapportnummer	:	FEL-93-B029
Titel	:	PRACTICAL: Planning en allocatie in C^2-domeinen met tijdkritische algoritmen
Auteur(s)	:	Drs.ing. C.W. d'Huy, ir. A.P. Keene
Instituut	:	Fysisch en Elektronisch Laboratorium TNO
Datum	:	februari 1993
HDO-opdrachtnummer	:	-
Nr. in IWP'93	:	704.2

Samenvatting (ongerubriceerd)

Planning is een even belangrijk als complex probleem in het bevelvoeringsdomein (Command and Control). Met een toenemende vraag naar snelle en flexibele oplossingen voor het toewijzen van middelen, het indelen van missies in een schema en het plannen van (grootschalige) operaties is het ondersteunen van menselijke planners met geautomatiseerde hulpmiddelen een noodzaak geworden. Herplannen is een belangrijke activiteit om de voortdurende veranderingen in de omgeving in het plan te kunnen betrekken. Daarnaast is het wenselijk om te kunnen anticiperen op opties in termen van "what-if" analyses. Dit rapport beschrijft een raamwerk voor (her)plannen gebaseerd op een scenario-gerichte aanpak met methoden en technieken voor het genereren van een plan, het herstellen van een plan en het (gedeeltelijk) hergebruiken van een bestaand plan. De basis voor het raamwerk is een conceptueel model van het proces van (her)plannen en een partitionering van kennis waarbij een onderscheid gemaakt wordt tussen strategische-, tactische-, causale- en wereldkennis. Een beheerssysteem voor temporele gegevensbanken wordt gebruikt om te kunnen redeneren over zowel het verleden, het heden en de toekomst. Het onderzoek heeft geresulteerd in het systeem PRACTICAL dat de toepasbaarheid van het raamwerk demonstreert, waarin de demonstratiesystemen CALIGULA an ALLOCATOR zijn opgenomen voor respectievelijk het alloceren van middelen en het toewijzen van piloten en vliegtuigen in een missieschema.

Contents

1 Introduction

People make plans in order to change them. In a Command and Control (C^2) domain detailed tactical plans are produced regularly, they are however destined to be radically changed during their execution as a result of changing requirements and changing resources **[Gadsden88]**. A second reason why plans should be reviewed is the fact that the planner has had to leave open spaces (or has had to make assumptions) in the plan due to incomplete and/or uncertain data. During execution or in the course of time the data used for the original plan may be changed, may be elaborated with more detail, or completely new data may become available. A third reason for plan review is that the world is not reacting to the measures as was projected by the plan. In **[Entin]** it is concluded that: "...even in the simplest C^2-environment, situation assessment and planning are dynamic processes. The dependent variables in one cycle become the independent variables of the next and the derived measures are closely interrelated."

Planning in a C^2-environment can be viewed as a dynamic and cyclic process:

1. An assessment of the situation;
2. A comparison of the current situation to the desired situation;
3. A comparison of the achievements of the current plan to the desired situation;
4. Making a selection of measures/options to achieve the desired situation or maintain the current situation.

One can view the C^2-planning paradigm as shown in figure 1.1. The initial tasks (goal tasks) are analysed and a set of measures is created. For this the planner uses a projection of these measures into the future to check if they meet the goal. When a successful plan has been created, it can be executed. The execution of the activities in the plan will be monitored to assure that the activities have the proper effect on the environment. When the monitoring does not find any deviation in the desired effects of the actions, execution of the plan continues. However, when the world changes in an undesired fashion, either because of an unwanted effect of an action (dependently/unforeseen) or because the world is changing independently of the plan, the execution of the plan is interrupted and diagnosis is started. The output of the diagnosis consists of a set of tasks that will undo the unwanted effects and will revise data for the existing plan.

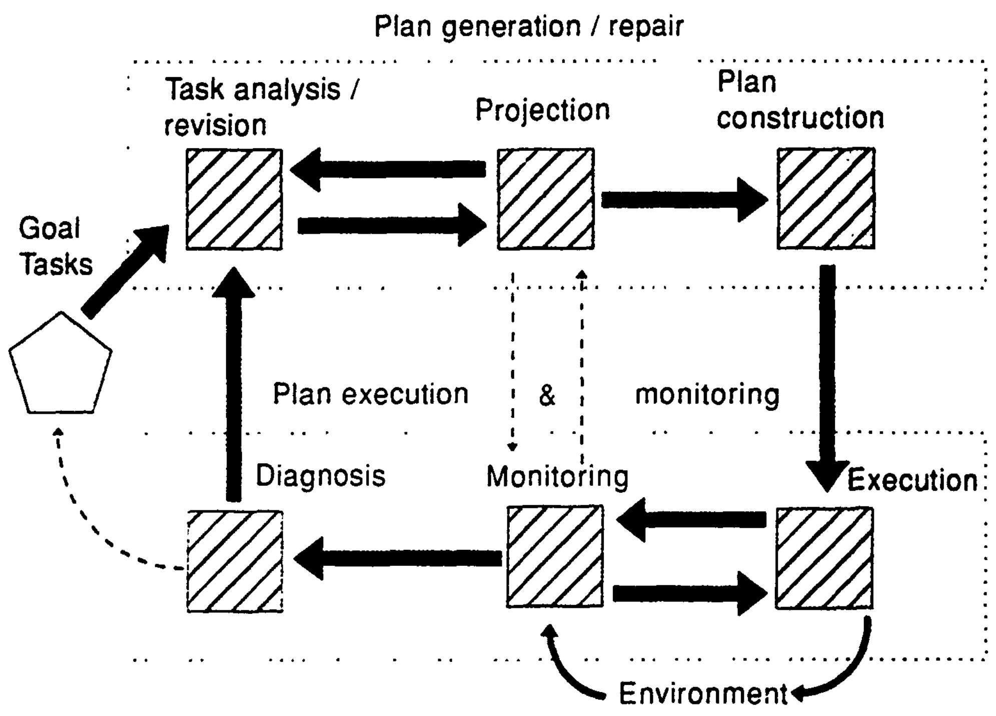


Figure 1.1: The C^2-planning paradigm.

Researchers in the field of Artificial Intelligence have put a lot of effort in defining frameworks and systems that are able to create a plan of actions out of a given current situation and a given desired situation of the world. Until now, reviewing or revising a current plan in such a way that it meets the requirements of the new situation has been done either manually or by planning all over again. We state that a human planning expert uses a specific kind of knowledge to adjust a previously created plan. This kind of knowledge can be used to define a framework that not only provides a way to specify knowledge to create plans, but that is also able to specify knowledge to revise a plan. In this framework both autonomous behaviour of plan generation, diagnosis and review as well as a certain degree of user interaction will be supported. Thus a planning system based on the specification defined by the framework can vary in its behaviour from a fully autonomous system to a system with a decision support character in which the user is in full control.

During the generation of a plan choices have to be made. For instance when several alternatives exist that all realise the same goal. One can view the evolution of a plan as the plan's history. We state that this history, together with the plan itself and the reason why the plan should be reviewed are the parameters of the plan review process. Therefore an important part of our research concentrates on means of analysing and using this history. It is plausible that the reason that a plan should be reviewed plays an important role in the reviewing and replanning process.

We have derived a framework for planning and replanning in C^2-processes that centres on methods and techniques for achieving plan repair or (partial) re-use. The basis is a conceptual model of the (re-)planning process in the context of Command and Control. In this framework a distinction is made between the different types of knowledge used in the planning process, aimed at achieving generic classes of task structures and rules. Another important distinction is made between knowledge used for plan generation and knowledge for plan repair. The framework is outlined in chapter 2.

Chapter 3 addresses temporal database management. In our view, planning necessitates a flexible approach to the notion of time in order to be able to reason about the past and present as well as the future. Moreover, a plan is typically developed from relative orderings between activities of which exact starting points and durations are not known beforehand. Therefore, incorporating only an absolute notion of time is insufficient, implying that a database enhanced with simple time stamping will not solve the planning problem. To this end, we use a Time Map Manager [Dean85] for temporal database management.

In the Command and Control context a plan is usually developed as an answer to a certain event or series of events taking place in the world. We use an approach based on scenarios and scripts to develop plans. This is a natural as well as flexible approach which allows the refinement of a plan via scripts containing goals to be fulfilled, actions to be undertaken to achieve these goals together with the allocation of resources to the actions. Moreover, it is a "piecemeal" approach to planning in the sense that it allows partial plan development that can be managed by the temporal database. This makes it possible to incorporate insufficient or uncertain data in partial plans that may be expanded at a later stage. This approach to planning is discussed in chapter 4.

Another point of interest is the use of algorithms to solve combinatorial problems. Since these are strongly related to resource allocation, an efficient method for solving these problems is important in a C^2-planning process. Again, means of keeping track of the solution path (history) are important when these algorithms are incorporated in the planning/replanning process. This is discussed in the context of resource allocation in chapter 5. Static resource allocation is addressed, whereby time plays no role other than a need for a speedy solution, and has resulted in the demonstrator CALIGULA for the problem of allocating frequencies to a radio link network. The problems in the domain of scheduling are dealt with, which has resulted in the demonstrator ALLOCATOR for the allocation of pilots to missions. Also, dynamic resource allocation during (re)planning is discussed here, specifically the problem of sharing limited resources by different activities.

Our research on (re)planning has resulted in the demonstrator PRACTICAL, based on the derived framework, temporal database management, planning via scripts and dynamic resource allocation. The demonstrator is described in chapter 6 and evaluated as to its mapping on the framework, the extent to which problems pertaining to (re)planning have been solved and its performance.

Chapter 7 contains some concluding remarks concerning potential bottlenecks for planning systems such as PRACTICAL, necessary further research, suggestions for an operational extension and applicability to C^2-processes. This document also contains a list of acronyms and abbreviations used, a glossary of terms and a list of references and suggestions for further reading. We have included data flow diagrams and information structure diagrams defining the conceptual model underlying the framework in appendices.

2 The framework

Primarily we distinguish between plan generation and the re-use or repair of a previously derived plan as these are the two main parts of the command and control *planning* process. Within these two parts we will focus on the specific kinds of expert knowledge used by human planners.

2.1 Plan generation

Generally speaking the plan generation process can be divided into four tasks:

1. Identify the next goal to be planned;
2. Collect sets of possible measures to accomplish this goal;
3. Select the most appropriate set of measures;
4. Incorporate this set into the current plan.

Experts use specific types of knowledge when performing these tasks. We will first distinguish between four major types, which will be specified in the next section. After specifying these four types of knowledge we will map them on the four tasks to be performed in the plan generation process.

2.1.1 Knowledge partitioning

We define the following types of planning knowledge:

1. *Strategic knowledge.*

   This type of knowledge guides the planning process in such a way that the most efficient solution path (performance of the system) and the most effective solution path (quality of the derived plan) are followed.
2. *Tactical knowledge.*

   This type of knowledge is best termed by: refinement knowledge. Tactical knowledge outlines a decomposition of the goal to be achieved into actions and sub-goals. Often more than one possible decomposition for the same goals exist, it is up to the strategic knowledge to choose one of them.

3. *Causal knowledge.*
   One should be able to predict the exact consequences of the actions proposed in the refinement for the current world. This requires causal knowledge.
4. *World knowledge.*
   World knowledge is the most basic type of planning knowledge and outlines what is known about the world at a certain time. Usually this type of knowledge is captured in a temporal database (see also chapter 3).

2.1.2 The use of knowledge in plan generation

After identifying the four tasks to be performed and the four types of knowledge which are used, we can map them on each other in the following way:

Task: identify the next goal.
Strategic knowledge. Since more than one (sub-)goal may exist, it is necessary to define the order in which they are handled. For instance this order may be given by a priority (importance of the goal) or via knowledge about expected side-effects of accomplishing the goal that implicitly implies the achievement of other goals (plan efficiency) **[d'Huy92a]**.

Task: collect sets of measures.
Tactical knowledge. One should collect all possible ways of achieving the selected goal (outlined in the tactical knowledge) to determine the 'best' course of actions.

Task: select set of measures.
Strategic knowledge. Using heuristics and/or rules it should be possible to determine the best set of measures to achieve the selected goal.

Task: incorporate measures.
After the goal is selected and the most appropriate decomposition into actions and sub-goals is derived these actions must be incorporated into the plan. Moreover the expected behaviour of the world to the proposed actions should also be captured to be able to determine proper goal achievements in the future. Incorporating the expected effects of the actions can also be very useful when a monitoring of the actions is required.
Tactical knowledge: expand the goal.

Causal Knowledge: determine the effects of the proposed actions given by the tactical knowledge.

World knowledge is used in every task to instantiate the other three types of knowledge.

2.2 Reviewing plans

As the world might not respond to the actions proposed in the plan as expected or when assumptions regarding the actions (e.g. their length) no longer hold, one has to review the plan which may lead to an adjustment of the new situation.

2.2.1 Reviewing methods

We distinguish two types of reviewing:

- *Plan repair.*
  Plan repair usually implies changing the values of variables in the tactical scheme and not so much changing the scheme itself (e.g. perform the same tasks by another agent because the original agent is no longer available).
- *Re-planning.*
  Re-planning as opposed to plan repair tries to change the tactical scheme by selecting other refinements for (sub-)goals and undoing the previous refinements. For instance if one wants to go to the airport one can travel either by car or by train. If during the execution of the plan too much traffic is encountered, one may choose to alter the tactic by leaving the car and taking the train.

It should be clear that in both cases we want to preserve and re-use as much as possible of the original plan.

2.2.2 The use of knowledge in plan review

As knowledge plays an important role in plan generation this applies also to the plan review process. The following types of knowledge have been identified:

- *Behaviour knowledge*
  Although one can predict the effect of an action in most cases, sometimes the world does not behave as expected. It is however possible to take these deviations into account via knowledge about what to do if the world responds to an action in another way than was expected. For instance if one wants to make some light in a room a default tactic could be to switch on the light. But if nothing happens and it is diagnosed (diagnostic module in the C^2-cycle) that the light bulb is faulty, the new tactic could be to first change the light bulb and then retry switching on the light.
- *Re-allocation knowledge*
  How should the planner react if a specific resource or agent that has been allocated to an action is not available anymore? Should it try to allocate another resource or agent from the same class or should the tactic be changed? In military operations planning the reason why the agent or resource isn't available anymore could be crucial and may lead to choosing another tactic instead of sacrificing more resources in a bad tactic.
- *Plan structure knowledge*
  It is important to be able to inspect the structure of the plan, i.e. which actions contribute to which goal and which sub-goals contribute to which more abstract (sub-)goal? Inspection of the structure is necessary to identify how the achievement of goals is affected by alterations in the execution of actions. For instance when an action takes a little more time than projected does this affect the goals we want to achieve?

3 Temporal database management

Representing time and reasoning about time plays a critical role in many facets of everyday problem solving. We continually have to make reference to what has happened, is happening, and might possible happen. To make matters more difficult, we have to cope with the fact that the world is constantly changing around us. To plan for the future we must be able to predict change, propose and commit to actions on the basis of these predictions, and notice when certain predictions are no longer warranted. All of this requires handling an enormous amount of complexly interdependent information [**Dean87b**]. This information is generally temporal in nature; predictions, plan choice and plan revision all require reasoning about facts occurring in time.

In contrast to storing facts in traditional data bases in which they remain 'true' until they are deleted, we like to relate facts to intervals of time in which they occur. Moreover we want to relate facts *relatively* according to the order in which they (should) occur. The latter requirement is one of the main reasons why simple time-stamping isn't sufficient.

3.1 TMM: the Time Map Manager

The Time Map Manager (TMM) designed by Dean is a comprehensive system for building a predicate calculus database of assertions that may be true over only certain intervals of time. The database is augmented with a mechanism for representing assertions that persist until contradicted (or 'clipped') and inference rules that assert new facts whenever particular conjunctions of facts are true at the same time. This database can be used to represent tasks and their effects and to dynamically order unordered tasks as the need arises.
Planning for a desired goal state by refining goals into less complex sub-goals and primitive atomic actions can be assisted by the TMM, since it maintains a map of time which contains the results of the refining process and adds a partial ordering to keep it consistent.

In [**Dean85, Dean87a, d'Huy92a**] the TMM is discussed in more detail, here we only discuss the basic entities in the TMM's database and some issues related to these.

3.1.1 Database entities

The data stored in traditional databases is best typed by *persistences*, as this kind of data persists (remains true) until stated different (e.g. deletion of data). On the other hand one likes to speak about data that is only true in a more or less known time interval (*events*). Both persistences and events are modelled in the temporal databases using the occurrence entity. An occurrence has three interesting attributes:

1. A fact (the assertion in a normal database);
2. A begin timepoint;
3. An end timepoint.

Via (partial) ordering of the timepoints, the occurrences are also (quasi) ordered **[d'Huy92a]**. This ordering can be made using the *distance* relation on timepoints of occurrences in the database. The distance relation is used to set a lower bound and an upper bound on the time between two timepoints.
Events are then modelled via an absolute ordering on the begin and end timepoint of the corresponding occurrence. Persistences are modelled using lower bound 0 and upper bound infinite.

3.1.2 Time map

The occurrences and the ordering on their timepoints can be depicted in a graph in which the distances are represented by edges and the timepoints by vertices. Such a representation is quite akin to maps like road maps and is therefore called a time map. In order to be able to explore these maps (e.g. to determine if something is true or false at a certain time) one needs a possibility to query the temporal database.

3.2 Temporal query interface language (Tequila)

There are two interesting query types:

- Determination of the order of timepoints (before, after, overlaps, etc.).
- Determination if a fact X is true throughout a specified interval given by two timepoints A,B.

The first query calculates the distance between the specified timepoints by searching paths (chains of given distances) connecting the timepoints. By adding the bounds of the elements in the path, the length of the path (also an interval) can be calculated. The length of every path between two timepoints gives an estimation for the actual distance. The calculation of the best estimation of a distance isn't that straightforward as it seems, as will be shown in the next paragraph.

The latter query type is in fact a composition of two queries of the former type as it can be modelled as follows:

1. Search for an occurrence *O* with a fact attribute equal to *X*;
2. Determine the distance between *A* and the start of *O*;
3. Determine the distance between the end of *O* and *B*;
4. If these distances have lower bounds equal or larger than zero then the query succeeds;
5. Otherwise, as long as there are still some other candidates repeat step 1 to 4.

3.3 Calculating path distances

As mentioned earlier the calculation of distances between timepoints is not as simple as it seems. The complicating factor is that the edges in a time map (the distances) are augmented with an interval in which the actual distance may vary. This means that when calculating the distance between two timepoints the lower and upper bound may be given via different paths. Consider for example the four timepoints *A*, *B*, *C*, *D* as in figure 3.1 below with the corresponding distances:

distance(*A*,*B*,[1,3]) (the distance between A and B lies between 1 and 3)
distance(*B*,*C*,[1,3]).
distance(*A*,*D*,[2,5]).
distance(*D*,*C*,[2,5]).

The lower bound for the distance from *A* to C is 4 given by the path via *D*. The upper bound is 6 given by the path via *B*. In other words when calculating the distance between two timepoints one searches for a path between these two with the highest lower bound, and for a path with the lowest upper bound.

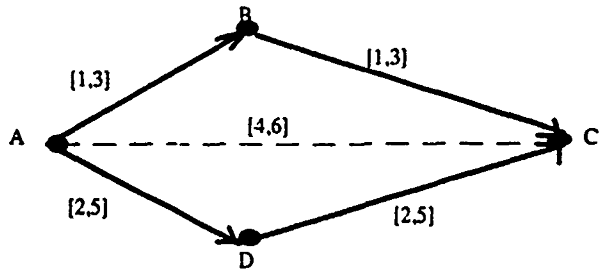


Figure 3.1: Path distance calculation example

Intuitively there are two approaches to calculating these distances:

- Searching for paths via a backtracking algorithm.
- Using a transitively closed time map.

It would be convenient if the time map is always transitively closed, since all the distances are obtainable at once. We belief however that due to the dynamic behaviour of C^2 processes (recalculation after altering a distance) the transitive closure is not a practical solution. The complexity of a reasonable good transitive closure algorithm from Floyd and Warshall is $O(n^3)$ [Aho75], with n=|N|, the number of nodes /timepoints. The impracticality is also confirmed by the observation that one does not need all the distances. As a matter of fact just a few at a time are necessary.

On the other hand, searching for the best fitting paths using backtracking has also a very unpleasant complexity. We have calculated the worst case complexity to be $O(n^{n-2})$. This is obtained as follows:

Def. 3.1:

P(k,n) is the number of possible (non cyclic) paths with length k, in a complete n-graph.

Def. 3.2:

P(n) is the number of possible (non cyclic) paths in a complete n-graph.

Corollary 3.1:

It is clear that from definitions 3.1 and 3.2 it can be deduced that:

$$P(n)=\sum_{k=1}^{n} P(k,n)$$

Note that P(1,n) = 1 for every n.

Theorem 3.1:

$$\forall k,n,1<k\leq n[P(k,n)=P(k-1,n)\times(n-k)].$$

Proof:

For a 2-graph it is clear. Consider an arbitrary complete graph with n nodes (n>2). Assign two points as resp. start (s) and finish (t). We are now going to construct non cyclic (s,t)-paths with length 2. Starting in s we can choose out of n-2 nodes to continue (n minus the points and t). From each of this chosen points we can finish in t. This leads to n-2 non cyclic (s,t)-paths with length 2. Thus P(2,n) = n-2. The same argument holds for constructing (s,t)-paths with length 3. After starting in s we can choose n-2 nodes for the next node in a path. From each of these nodes one can choose out of n-3 nodes to continue. This leads to P(3,n)=(n-2)(n-3). In general P(k,n)=(n-2)(n-3)....(n-k) and hence P(k,n) = P(k-1,n)*(n-k).

Theorem 3.2:

The calculation of a distance via searching paths runs in $O(n^{n-2})$.

Proof:

From corollary 3.1 it follows that:

$$P(n)=\sum_{k=1}^{n}[P(k,n)].$$

Using theorem 3.1 this can be written in full as follows:

1

1*(n-2)

1*(n-2)*(n-3)

1*(n-2)*(n-3)*(n-4)

.

.

1*(n-2)*(n-3)*(n-4)* ... *(n-n)

-------------------------------- +

$\alpha n^{n-2} + \beta n^{n-3} + \gamma n^{n-4} + \ldots$

As αn^{n-2} contributes the most to this expression it easily shown that the complexity indeed equals: $O(n^{n-2})$.

One can further derive a binomial notation for P(n) from the above calculation:

Corollary 3.2:

$$P(n)=\sum_{k=1}^{n-1}\frac{(n-2)!}{(n-k-1)!}$$

3.4 Implementation approaches

We have implemented the TMM in Quintus Prolog and used Prolog's predicate database to model the temporal database. We have defined the following facts:

```
occurrence(occ_id,fact).
distance(point1,point2,low,high).
```

In the distance facts the point arguments typically refer to the beginning or end of an occurrence (e.g. begin(occ1)).

3.4.1 Heuristic search in path distance calculation

As a first step towards finding an efficient path distance calculation algorithm we first implemented an algorithm that aggregates all possible paths between two given points. This algorithm in its basic form resembles the algorithm described above which has the bad complexity of $O(n^{(n-2)})$.

This can be improved by taking advantage of the structure of the time map. It is often sufficient to determine only the order of two occurrences instead of the precise distance. Therefore one can stop searching paths when a path has been found in which the upper bound and the lower bound have the same size. Furthermore one can use heuristics to guide such a search. We have identified the following heuristics, when only searching for the order of two occurrences:

- Does a direct distance reference exist?
- Traverse a goal before the goal's refinement.
  This can speed up the algorithm, since the length obtained by the refinement of the goal has to lie within the bounds of the length defined for the goal.
- Traverse from a beginpoint of an occurrence to the endpoint of that occurrence, before switching to another occurrence.
  This leads to a preference for paths with alternating occurrences (e.g. begin(1), end(1), begin(2), end(2), etc.). Since the distance set *on* occurrences generally contributes more to the length of a path than distances *between* occurrences, a good estimate of the actual distance should be found with less effort.

When a best estimate of a distance is required only one heuristic has been identified:

- Traverse a goal's refinement instead of the goal itself. In other words the algorithm should only traverse a goal if a refinement doesn't exist. Since a refinement has to lie between the bounds of the goal, the path found using the refinement is always the same or better than the one obtained by only traversing the goal.

Applying the heuristics that have been identified results in a considerable increase of performance. But when querying large temporal databases, the heuristics guided aggregation algorithm still lacks the performance necessary for practical use.

3.4.2 Ford's Algorithm

It is clear that caching earlier calculated distances can contribute highly to the performance of the calculation of distances. Therefore we have used Ford's algorithm [**Even79**] to calculate the shortest paths (for upper bounds) and longest paths (for lower bounds) to all other points starting from a unique point called 'ref'. These path lengths are stored in so called **lambda facts** (*lambda_min, lambda_max*) After this each path between two arbitrary points can be easily calculated by subtracting the appropriate lambda values. Ford's algorithm is as follows:

```
1.  $\lambda(ref) := 0.$
2.  For all timepoints: $t \neq ref, \lambda(t) \neq$ infinite.
3.  As long as there is an edge $e{:}x \rightarrow y$ such that
    $\lambda(y) > \lambda(x) + length(e)$ do
    $\lambda(y) := \lambda(x) + length(e).$
```

The algorithm actually calculates the shortest (or longest) paths to all vertices (timepoints) starting in 'ref'. It allows edges to be negative (Dijkstra's algorithm doesn't, see [**Even79**]). On the other hand it does not allow a directed circuit whose length is negative, however, this is something that should indeed not occur in a time map due to the properties of time! If a negative circuit exists it will be detected however. Ford's algorithm runs in complexity O(ne), with n the number of nodes/vertices and e the number of edges. As time maps tend to be sparse (we have identified an average adjacency degree of 3) this complexity is far better than the $O(n^3)$ obtained by the transitive closure.

Beside the more pleasant complexity of Ford's algorithm there is another advantage in using this algorithm. As intermediate distances are cached in the lambda's, it is possible to calculate more distances using the same lambda's, as long as the time map isn't altered.

3.4.3 Computational framework for organising temporal facts

The most expensive operation distinguishing temporal data bases manipulations from those performed by static database systems involves finding occurrences that satisfy certain temporal constraints. This operation, which is called *token-* or *occurrence retrieval* is the temporal analogue of fetching assertions from the database that match a given pattern. Token retrieval requires the database to search for occurrences whose type matches a given pattern and whose associated interval spans a specified reference interval. Thomas Dean [**Dean87b**] remarks in his paper on large-scale temporal databases: "As our representations have become more sophisticated and our ambitions to tackle more realistic domains have grown, the problems inherent in managing large temporal databases have become a major factor limiting growth".

Using algorithms like Ford's contributes highly to the system's performance, but even then the management of databases with several hundreds or thousands of occurrences becomes a problem. Dean proposes a dual-indexing mechanism in addition to the quite optimum algorithms like Ford's.

First, it shouldn't be necessary to recall (i.e. search through) all of the events in the past, present and future. Only the most recent ones are likely to be of interest. Restricting attention to a particular time frame requires that facts that change over time are *indexed temporally*, that is to say, stored in such a way that facts and events common to a given time frame are easily accessible from one another. One could for instance use a partitioning of the time map in parts corresponding to: weeks, days, eight-hour work shifts, one-hour intervals etc. Such a partitioning could be implemented using a discrimination tree (DTREE) as described in [**Charniak87**].

Secondly, one would also like to avoid expending energy on facts that have nothing to do with the specific matter one is dealing with. This requires that facts be *indexed semantically*. To accomplish this index one has to provide *context dependent* knowledge in order to discriminate between facts.

As with any scheme for speeding up retrieval, there is a cost associated with organising the data to support these fast retrieval routines. Fortunately, much of the work required for organising occurrences is already handled by the basic routines that manage a time map.

4 Knowledge-based (re)planning

In this chapter we will describe how the framework defined in chapter 2 is used for the tasks within planning and replanning. The role of temporal database management as outlined in the previous chapter will be clarified. To illustrate the planning and replanning strategies we will use an example from the domain of fire contingency planning.

4.1 The planning strategy

4.1.1 Scenarios and scripts

An important aspect of C^2-planning as opposed to classical planning is its *threat → countermeasure* nature. Each C^2-planning problem can be described in terms of a mission (e.g. maintain current state, achieve a certain state) and more or less aggressive actions of an adversary force to this mission (e.g. enemy forces, the behaviour of an unstable chemical process). The planning strategy that we have adopted in the context of PRACTICAL is based on the assumption that these possible adversary actions can be determined beforehand (e.g. enemy doctrine, analysis of the chemical process) and described in scenarios. Moreover one can also determine possible countermeasures in order to deal with each scenario. These countermeasures can be described using scripts.

A scenario can be described in the following terms:

Who: The competitor expected to pose the threat (e.g. 4th Armoured Division or heater II).

What: The nature of the adversary initiative and its assumed objective (e.g. crossing border in order to capture Braunschweig or burning too high in order to overheat kettle II).

Whom: Which of 'our' organisation's units are targeted (e.g. 90th Mech. Infantry Division or Kettle II).

How: Lays out the competitor's likely plan of actions (e.g. crossing border, force our forces to cross the Weser river, take positions in the Harz mountains, surround Braunschweig, take Braunschweig; heater burning at full power too long such that kettle reaches a critical temperature).

When: Details the conditions that might apply (e.g. the weather should be good in order to allow enemy tanks to traverse sector X; conditions during the night when no human operator is at hand to operate the heater).

Generally scripts can be defined over the same dimensions as scenarios; the focus of initiative however rest on units/objects in our control. Scripts are usually described in terms of (sub)*goals* that have to be fulfilled and *actions* that have to be carried out. The (sub)goals in the script will in turn be refined in other scripts. Thus the scripts reflect the projected course of actions.

We will illustrate this with the following simple generic script defining counter measures to take when dealing with a scenario in which a building is on fire. Variables are denoted in capitals. Instantiations of these variables will result in actual scripts as instantiations of the generic scripts.

```
script(fight_fire(Fire,Size,Location,Casualties)) ->
        achieve(allocate_teams(Fire,Size,Teams)), then do
        action(goto_fire(Fire,Teams,Location)), do
        achieve(handle_casualties(Fire,Location,Casualties)) and
        achieve(determine_method(Fire,Method)), then do
        achieve(extinguish_fire(Fire,Location,Method)).
```

A scenario has a corresponding top-level script with a top-level goal, in the above example the latter would be `fight_fire(Fire, Size, Location, Casualties)`. There may possibly be strategic options amongst the scripts, of which the subgoals will be refined in other scripts, also with possible strategic options amongst them. The corresponding scripts are typically detailed, we will illustrate this later in this chapter after introducing terminology and script refinement notions.

A scenario is triggered by one or more *world events*. The world event for the above example would be the fire alarm `fire(Fire, Location, Size, Casualties)`. Compositions of world events thus serve as *indicators* for scenarios **[Sutherland90]** which describe the *expected* course of actions. The hierarchy of generic scripts describe actions that are in effect an *answer* to the expected course of the competitor's actions. This approach thus provides possibilities for red-blue-white force scripting. The implementation of indicators is achieved by a rule base containing *event rules*. The event rules may indicate several scripts to be applicable for dealing with the world event. This implies a strategic option for the selection of a script. The choice of script is governed by what has been defined in the framework as strategic knowledge. The implementation of this type of knowledge for script options is achieved by *script selection rules*. For instance, in the example above the determination of the fire-fighting method is an example of the application of strategic knowledge. A default script selection rule for the script that refines the goal `extinguish_fire(Fire, Location, Method)` may take the method to be `water`. Other scripts for the same top-goal may for instance use methods `foam`, `halon` and `spray`.

A second instance of strategic knowledge is *goal selection*. Given a script, there may be occasions that the various subgoals in the script may have to be executed in a different order than the order (typically chronological with respect to start time) in which they are specified in the script. This type of knowledge is implemented by *goal selection rules*.

4.1.2 Goal refinement

By the refinement of a goal we mean the way in which the goal can be achieved. There may be various options, as for instance in the example above for the goal: `extinguish_fire(Fire, Location, Method)`. Refinements are governed by what we have named tactical knowledge. The distinction between the various methods for the refinement of the above goal is an example of tactical knowledge. To be clear, the *choice* between the options is, as stated above, an example of strategic knowledge. The expansion of the goals in scripts is again governed by tactical knowledge.

We use the following basic functions in the scripts for instantiating occurrences, goals and distances in the temporal database:

```
insert_occurrence(Context, Parent_GoalID, OccurrenceID, Occurrence)
insert_goal(Context, Parent_GoalID, GoalID, Goal)
insert_distance(distance(Context, Begin, End, Low, High))
```

Prior to each goal expansion a context is created in which the current plan state is saved. This is used for keeping track of the plan evolution in order to be able to restore a certain context at a later stage. On inserting occurrences and goals in the temporal database the parent goal (see section 4.2) is registered as well. This enables tracking the goal that was being expanded at this stage. Both these mechanisms are necessary for replanning, which will be addressed in the next section.

The insertion of a new goal is accompanied by the placing of the goal on the *goal agenda*. Ideally, each goal will have a corresponding script containing the (tactical) knowledge about how the goal is refined (expanded) to achieve its fulfilment. We note however that it may be so that a goal does not have a corresponding script (incomplete knowledge). Plan evolution will continue until there are no more goals on the agenda.

Each insertion of an occurrence or a goal will be accompanied by insertions of distances in the temporal database. The distances have a begin- and an end timepoint with a lower and upper bound. If the distance is known exactly, the latter two will coincide. Typically, three distances will accompany each occurrence or goal X, namely the distance from a previous timepoint to begin(X), the distance from begin(X) to end(X), and the distance from end(X) to another timepoint. This is illustrated in figure 4.1 below:

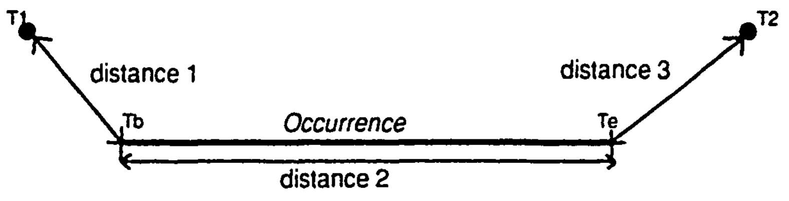


Figure 4.1: Example of occurrence insertion

4.1.3 Constraints

An additional instance of tactical knowledge is the application of constraints in the scripts. For instance, the *precondition* that there has to be a fire team available that can be allocated to the fire at hand. The allocation of resources in scripts uses the temporal database to find appropriate resources, restricted by the constraints that 1) appropriate resources are available (for instance: one or more fire teams), and 2) that they are available in the required *time window* (for instance: during the complete predicted duration of the achievement of the goal). This feature uses the temporal query interface language. An example is the following:

```
findall
    (true_throughout(Context,at_location(Team, Place), begin(Goal), end(Goal))
     & not (occurrence(Context, OccupiedID, occupied(Team))
        &  overlaps(Context, OccupiedID, Goal) )  ).
```

The above procedure will search the temporal database for those fire teams that are at a certain location throughout the duration of the required goal (e.g. fight fire) and that are not occupied during a time window overlapping the goal. The latter constraint is necessary because the team(s) may be allocated to other goals prior to the required goal or in some time in the future.

The notion of *occupied* used in the example above is necessary to be able to keep track of allocations. Another feature necessary for maintaining allocations is the so-called *protection*, in effect a constraint on possible allocations. For instance, during a goal `extinguish_fire_at(Location, Fire, Team))` we would like to have the team to remain at the location. A protection will be used in this case to ensure that the team is not allocated to another goal during the time window in which the goal is being fulfilled.

Apart from *availability* constraints such as illustrated above constraints could be applied concerning *necessity* and *sufficiency*. These types of constraints will restrict the type of resource instance required (e.g. a specific fire team trained in chemical fire fighting), or sufficient resource instances, possibly as relaxations of the necessity constraints in the case these can not be met. The subject of resource allocation will be discussed extensively in the next chapter.

4.1.4 Maintaining consistency

The expansion of goals in the respective scripts ultimately results in a *scheme of actions* that will in effect be the plan. The actions will have corresponding time windows with start and end timepoints and predicted durations, all expressed in distances with lower and upper bounds. A problem is posed by the *effects* of actions on the state of the world as modelled in the temporal database. The problem is that simply asserting the actions in the temporal database will in general cause the database to become inconsistent. The reason is that facts, and specifically temporal facts, are context dependent.

Consider the occurrence `at_location(team1, fire_station)`. An action `goto_fire(team1,from(fire_station), to(hospital))`, when executed in the world, will necessarily result in the occurrence `at_location(team1, hospital)` becoming true. This implies positive effects of actions in the sense that new facts are *added* to the existing world knowledge. Likewise, there is a negative effect as well in the sense that facts previously true now no longer hold. In this example, the occurrence `at_location(team1, fire_station)` is evidently no longer valid. Thus, the occurrence will have to be *clipped*, implying that the end time of the occurrence is adapted such that the occurrence is true until the start time of the occurrence `at_location(team1, hospital)`. Note that occurrences in the context of a temporal database will never be deleted in principle, the notion of clipping will always retain the temporal history. Maintaining this type of consistency is known as the *frame problem* **[Hayes69]**. Essentially, the frame problem is to be able to keep track of what is and isn't true at a certain time at a certain place. One can see this as a matching problem between different frames of a motion picture.

A second problem is keeping track of more implicit consequences of actions. To illustrate this notion, let's assume that we keep track of the locations of the fire trucks used by fire teams as well. In the case that a team moves to another location not only should "frame" adaptations be made in the database as outlined above, but also an implicit consequence such as the new fact that the fire truck(s) used by a fire team are now known to be at the new location as well. This is known as the *ramification problem* **[Finger87, Dean91]**.

To solve the frame and ramification problems we have implemented *adder* and *clipper* rules as an instantiation of what has been discerned in the framework as causal knowledge. The accompanying procedure `add_effect` handles adders and clippers. For adders this is quite straightforward. With the addition of an occurrence two distances will be inserted in the temporal database. The first distance has (0,0) bounds between the end of the occurrence causing the effect (e.g. `goto_fire(team1, from(fire_station), to(hospital))`) and the beginning of the effect. The second distance will instantiate the effect itself as a persistence in the temporal database, thus with lower bound 0 and upper bound infinite.

The implementation of the analogue for clippers is less trivial. This entails a path distance calculation to determine the resulting distance from the beginning to the end of the occurrence that is clipped. In effect, this is done by calculating the distance from the reference point (ref) to the beginning of the effect and the distance from ref to the beginning of the occurrence to be clipped and calculating the difference[1]. This is illustrated in figure 4.2 below. This is in principle

[1] Note that the difference may be negative, in which case the occurrence starting earlier is clipped. Hypothetically, the situation may occur that the starting points of the effect and the occurrence to be clipped coincide due to the latter having bounds (0,0). This can be avoided by assuming facts to always have a duration of at least one time unit and thus a lower (and upper) bound of at least 1.

a costly operation, we note however that with Ford's algorithm as outlined in chapter 3 the lambda's derived by the algorithm can be used, acquiring a considerable increase of performance up to instantaneous if the lambda's are still valid (time map not yet altered prior to clipping).

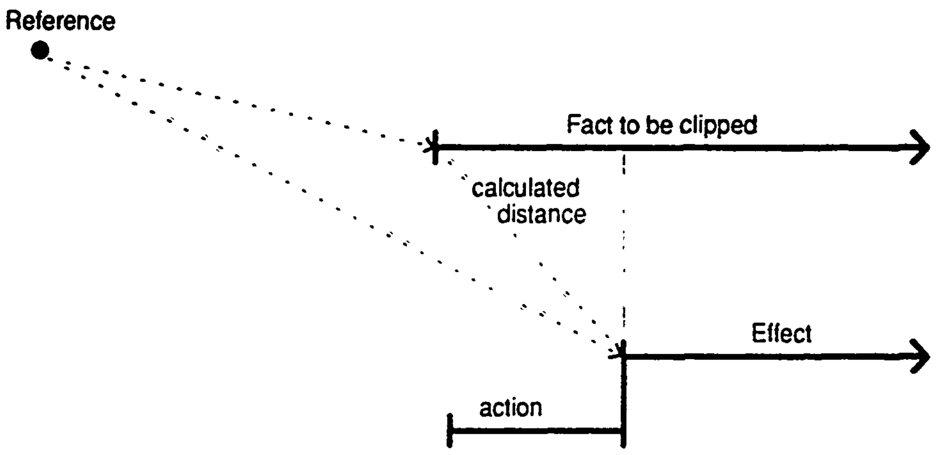


Figure 4.2: Illustration of the clipping procedure

The complete top-level script for the simple fight_fire procedure used in this section for a medium size fire is now as follows:

```
script(GoalID, fight_fire(Fire, medium, Location, Casualties)) ->
   occurrence(Context, EventID, event(fire(Fire, Location,medium,Casualties)),
   findall(
   true_throughout(Context,at_location(Team,Place),begin(GoalID),end(GoalID),
    not (
       occurrence(Context,OccupiedID,occupied(Team)),
       overlaps(Context,OccupiedID,GoalID)
       )
   ),
   insert_occurrence(Context,GoalID,Action1,
     action(goto_fire(Team,from(Place),to(Location)))),
   insert_distance(distance(Context,begin(GoalID),begin(Action1),5,5)),
   insert_distance(distance(Context,begin(Action1),end(Action1),10,15)),
   insert_occurrence(Context,Action1,OccupID,occupied(Team,Fire)),
insert_distance(distance(Context,begin(GoalID),begin(OccupID),0,0)),
insert_distance(distance(Context,end(OccupID),end(GoalID),0,0)),
distance(Context,begin(GoalID), end(GoalID),Min,Max),
insert_distance(distance(Context,begin(OccupID),end(OccupID),Min,Max)),
insert_goal(Context,GoalID,Goal1,extinguish_fire_at(Location,Fire,Team)),
insert_distance(distance(Context,end(Action1),begin(Goal1),5,8)),
insert_distance(distance(Context,begin(Goal1),end(Goal1),200,400)),
insert_distance(distance(Context,end(Goal1),end(GoalID),0,0)),
add_effect(Context,Action1,
   action(goto_fire(Team,from(Place),to(Location)))).
```

4.2 The replanning strategy

Why should one review a previously derived plan? We have identified three main reasons:

- An allocated resource is not available anymore:
- The bounds of one or more actions exceed those set in the plan:
- The chosen tactic does not live up to the expectations.

The first reason will be handled in section 5.3 in which allocation issues are discussed.

The second reason is one of the most basic plan monitoring problems. i.e. are the actions executed within their preset bounds? When an action exceeds its upper bound several things may happen. Below we recite three cases and describe how we have handled them.

- The action may interact undesirably with other actions because they use the same resource. We propose a quite straightforward algorithm to solve these problems by searching pairs of action occurrences that use the same resource. Next an overlap query is performed on these two occurrences. If an overlap exists a different resource can be allocated to either of them following the approach described in section 5.3.
- The action may interact undesirably with other actions because they cannot be performed at the same time. Such pairs of action occurrences can be found easily using rules which identify conflicting actions (an analogue of the clippers). Next an overlap query is performed on these two occurrences. If an overlap exists then it is up to the operator/user to 'solve' the interaction (e.g. postpone one of the actions or try another tactic).

- Although no undesirable interactions occur, goals may not be achieved within their estimated bounds. We have used an alternate structure on the time map in which the hierarchy of parent goals → subgoals → actions is more clear. Using this hierarchy it is quite simple to identify goals that cannot be achieved within their estimated bounds. The detection can be done using the following algorithm which uses two facts that implement the alternative structure (`action_parent( action_occ_id, goal_occ_id)` and `goal_parent(goal_occ_id,parent_goal_occ_id)`). The algorithm starts at the action and locates its parent goal. If this goal has to be expanded in order to capture the new bounds of its child action, the user is warned and asked for a confirmation. After this the parent of this goal is located in order to check if this one has to be expanded in order to capture the new bounds of its child goal. This location/confirmation procedure is repeated until no further expansion is required. Expansion of the bounds of a goal might imply the postponement of other goals, hence each of the other (replanning) conflicts mentioned in this section might occur.

It is clear that replanning as opposed to planning is directed at supporting the user in replanning decisions because it is very difficult to model all considerations a user normally makes in replanning issues. What it can do however is support the user by presenting the effects of the replanning choices, therefore a good man-machine interface is required.

An example of a more intelligent support is the following in which the user asks the system to try a different tactic for a goal ("what-if" analysis). In order to incorporate this the following is done:

- Save the current context (all occurrences, distances, the goal agenda);
- Remove subgoals with actions, action effects and protections of the culprit goal. This also implies a backward evaluation of the clipper and adder rules in order to re-install the effects that have been affected by actions in the culprit goal's plan;
- Plan the culprit goal again using a new tactic.

The replanning strategy described above results in a partial plan deletion. However, we have minimised the deletion due to locating the culprit goal and explicitly replanning this goal only. Thus, a large part of the plan is kept and only the relevant defect portions are planned again. The context saving mechanism is used in the strategy. Context saving has a low computational overhead, does however require memory space. This poses no problems, contexts can be saved in files or even in easily accessible secondary storage means (e.g. CD-ROM) if necessary.

An alternative for a context saving mechanism would be to use for instance an ATMS - Assumption-based Truth Maintenance System - to keep track of assumptions and implications. In the case of an assumption violation (e.g. in our application a planned goal with its expansion that has to be replanned) all the involved assumptions and their implications (goal expansion results) will have to be deleted and inserted into the ATMS again during replanning. The computational overhead is thus extensive [**Kleer86c, Keene91**]. Another option is so-called ***case-based planning***. This approach centres on (re)planning by making use of previous plans that are similar to the plan at hand [**Hammond90**]. This does however imply an extensive effort to acquire effective matching algorithms (and possibly learning algorithms as well) for locating a plan that can be taken as a basis. This is somewhat similar in nature to a *templating* approach which we have implemented by way of scenarios and scripts. However, as described extensively in this chapter, the script-based approach is far more flexible because portions of plans are developed, allowing an incremental plan development from generic scripts, instead of developing a plan with an actual existing similar plan as a basis.

5 Resource allocation

5.1 Static resource allocation

The allocation of resources is a major constituent of scheduling and planning systems. In these cases the factor time is a central focus point. In this chapter we will discuss resource allocation for scheduling and planning in sections 5.2 and 5.3 respectively. We will first discuss resource allocation in a static context where time is not incorporated other than as a requirement that the allocation must be acquired as fast as possible, and allocations may have to be recalculated over some period of time. Examples of static resource allocation are the map colouring problem, the so-called "travelling salesman" problem (finding an optimal route) and the allocation of frequencies to links in radio nets[2].

5.1.1 General CSP formulation

The general formulation of a static resource allocation problem is the following: we are given a set of nodes, a finite and discrete domain of possible values for each node, and a set of constraints. A constraint is defined over some subset of the set of nodes and specifies a requirement on some combination of values for that subset of nodes. An edge is a binary relation between two nodes: given n nodes, there may be at most (n over 2) edges, the number of ways binary selections can be made from n objects. A binary constraint is a constraint on two nodes, thus a constraint on an edge. For simplicity, we assume binary constraints.

We will formalise the above notions to be able to discuss complexity and thus efficiency issues in this chapter. A formal definition of such a *constraint satisfaction problem* (CSP) is then as follows [Mackworth77, Henderson86, Kumar92]:

2 This latter example has in fact been chosen as the benchmark problem for the EUCLID RTP 6.4 project on combinatorial algorithms for military applications.

$N = \{i,j,k,...\}$ is the set of nodes, with $|N| = n$; [3]

$V = \{a,b,c,...\}$ is the domain of all possible values, with $|V| = v$;

$V_i = \{d,e,f,...\}$ is the domain of possible values for node i, with $|V_i| = v_i$;

$E = \{(i,j) \mid (i,j)$ is an edge in $N \times N\}$, with $|E| = e$;

R_1 is a unary relation, (i,a) is admissable if R_1 (i,a);

R_2 is a binary relation, (i,a)-(j,b) is admissable if R_2 (i,a,j,b).

The constraints can be expressed in terms of admissability relations. The solution to a CSP is now the set of all n-tuples in V^n that satisfies the given relations.

5.1.2 Algorithms for CSP

Solving constraint satisfaction problems such as formalised above can be done in several ways. The first method one could try is to randomly assign values to the set of nodes. Hereafter the set of constraints (in terms of admissibility relations) is run through to see if the assignment satisfies all constraints. Evidently, this will typically not be the case. Moreover, the algorithm is in principle non-terminating, because the assignments of values may be duplicated or assignments may not be achieved at all. If a feasible assignment is achieved, it is due to sheer dumb luck.

Another approach is the so-called "generate-and-test" algorithm, which differs from the above only in that it generates all possible combinations incrementally instead of in a random fashion. It could be implemented by for instance a backtracking algorithm that successively runs through all the possible combinations in a depth-first search, checking the satisfaction of the constraints after each generation of a value assignment for all the nodes. A first solution will suffice and will in principle be found sooner than by the random generation method outlined above. The worst case complexity of the generate-and-test algorithm would be of $O(ev^n)$, because for every node all values in the domain V (assuming all nodes to have domain V) will have to be assigned in the end, and for every assignment all the edges e will have to be checked for constraints on them. Given 100 values, 1000 nodes and thus in worst case (1000 over 2) = 1000*999/2 edges (the number of binary combinations of 1000 nodes), this leaves us with a dazzling complexity in the order of 10^{2005}. Thus the generate-and-test algorithm is a "brute force" approach.

3 We assume a knowledge of some set theoretic notions here. The notation $|N| = n$ is used for the *cardinality* of a set N, the number of elements in N. A relation is a value-mapping on a set of variables, here an assignment of value(s) to one and two nodes respectively for a unary and binary relation.

There are numerous algorithms that refine the backtracking mechanism by pruning possible paths in order to eliminate search that is known to be futile in advance. Examples are hill climbing, min/max-search and the A*-algorithm (similar to branch-and-bound) [**Aho75**]. Expanding on the backtracking techniques, there are forms of intelligent backtracking such as dependency directed backtracking [**Stallman77**] and intelligent backtracking schemes [**Kumar88, Bruynooghe81, Bruynooghe84**]. Dependency directed backtracking is the basis for truth maintenance systems such as the TMS of Doyle [**Doyle79**] and the ATMS of J. de Kleer [**Kleer86abc**]. Also, techniques used in operations research such as linear and dynamic programming and simulated annealing could be applied [**Fox89, d'Huy92a**]. One could even approach solutions with genetic algorithms, see for instance [**Nygard92**].

There is however a major drawback to most of the above mentioned approaches: the constraints are checked only after a combination of values for the nodes or a partial combination has been selected. Also, no memory is kept as to the values that are no longer valid for a node due to inconsistency with the constraints. This does not apply to the ATMS, the latter however generates an enormous overhead in keeping track of assumptions, their implications and the consequences of values becoming invalid [**Keene91**].

5.1.3 Constraint propagation

The most powerful and efficient method to date for solving (static and well-formed) constraint satisfaction problems is by using *constraint propagation* [**Kumar92, Henderson86, Kleer89**]. The basic algorithm may roughly be paraphrased as follows:

1. Initialise nodes by assigning value domains (global and current, initially equal) for each node, and the constraints;
2. Select the best next node X (according to certain criteria);
3. Assign a value to node X from its global value domain (according to certain criteria);
4. Adapt the value domain of X, resulting in a current value domain containing only the assigned value;
5. Apply arc consistency (explained later) to node X, resulting in the adaptation of current value domains of other nodes;

6. If any of the value domains become empty, the assignment of the value to node X is inconsistent and it is retracted from the global domain of X, then continue from step 3 onwards;
7. If step 6 does not apply, continue from step 2 onwards until a solution is found (the current value domains all contain one element).

The current value domains are thus initially equal to the global value domains, but are decreased during the process, resulting in singleton current value domains if a solution exists at all. The propagation of constraints is achieved by the *arc consistency* mechanism. Arc consistency is applied to the constraints restricting the value pairs of two nodes, thus the constraints on edges of a graph. If we have a node X with domain [a] and a node Y with domain [a,b] and a constraint on edge (X,Y) stating that the values of X and Y may not be the same, arc consistency applied to edge (X,Y) will result in the elimination of value *a* from the domain of node Y.

An edge (X,Y) is arc consistent if for every value a in the current domain of X there is some value b in the current domain of Y such that X=a and Y=b are admissible by the binary constraints on edge (X,Y). Arc consistency is *directional*; if an edge (X,Y) is arc consistent, it will not necessarily be the case that edge (Y,X) is arc consistent as well.

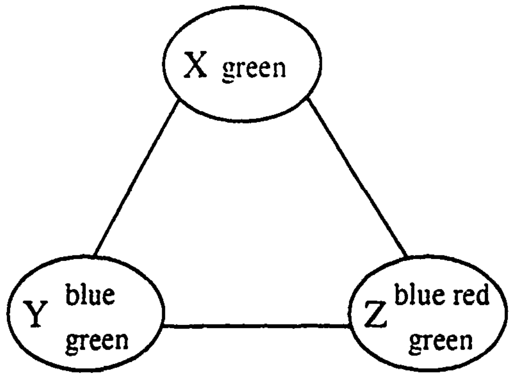


Figure 5.1: Arc consistency example

We will further illustrate arc consistency by a simple map colouring problem with three nodes X, Y and Z with respective domains [green], [blue,green], and [blue,red,green], see figure 5.1 above. The constraints on all edges are that no connected nodes may have the same colour. Applying arc consistency to the domain of node X results in the elimination of value green from both the domain of Y and the domain of Z. The domains of X and Y are now singletons (containing green and blue respectively), node Z still contains red and blue. Applying arc consistency to node Z will eliminate the value blue, resulting in the respective assignments green, blue and red to nodes X, Y and Z. In this example a single solution is found, in general there may be no solution or more than one solution.

The complexity of an efficient constraint propagation algorithm is of $O(ev^2)$ **[Henderson86]**. This implies an enormous gain of $O(v^{n-2})$ in comparison with the brute force generate-and-test approach. For the example in paragraph 5.1.1 above, the complexity of $O(10^{2005})$ is decreased to $O(10^9)$ (!).

5.1.4 CALIGULA

We have implemented a program that uses constraint propagation to solve CSP for the case of frequency allocations to a radio link network. We have named it CALIGULA - Combinatorial ALgorIthms GUiding radio Link frequency Allocation [4]. The program is incorporated in the PRACTICAL demonstrator and written in Quintus Prolog **[Prolog]**.

Reviewing the algorithm for constraint satisfaction outlined above, it is necessary to have a mechanism that can "backtrack" to previous states in the case an assignment of a value to a node turns out to be inconsistent with the constraint propagation to other nodes. To solve this, we did not use actual backtracking but a context saving mechanism. When a value is assigned to a node, the complete state of global and current domains and assigned frequencies is saved in a ***context***. When an inconsistency arises, the latest context - prior to assigning the culprit value - is restored and the current domain of the corresponding node is set to its global domain from which the culprit value is deleted.

The constraint propagation process will start with a certain node. For this start node domain values may be found that are inconsistent with the constraints. These values can therefore never be assigned to that node. After constraint propagation has found a solution, these inconsistent values will have been deleted from the *global* domain of the start node during propagation.

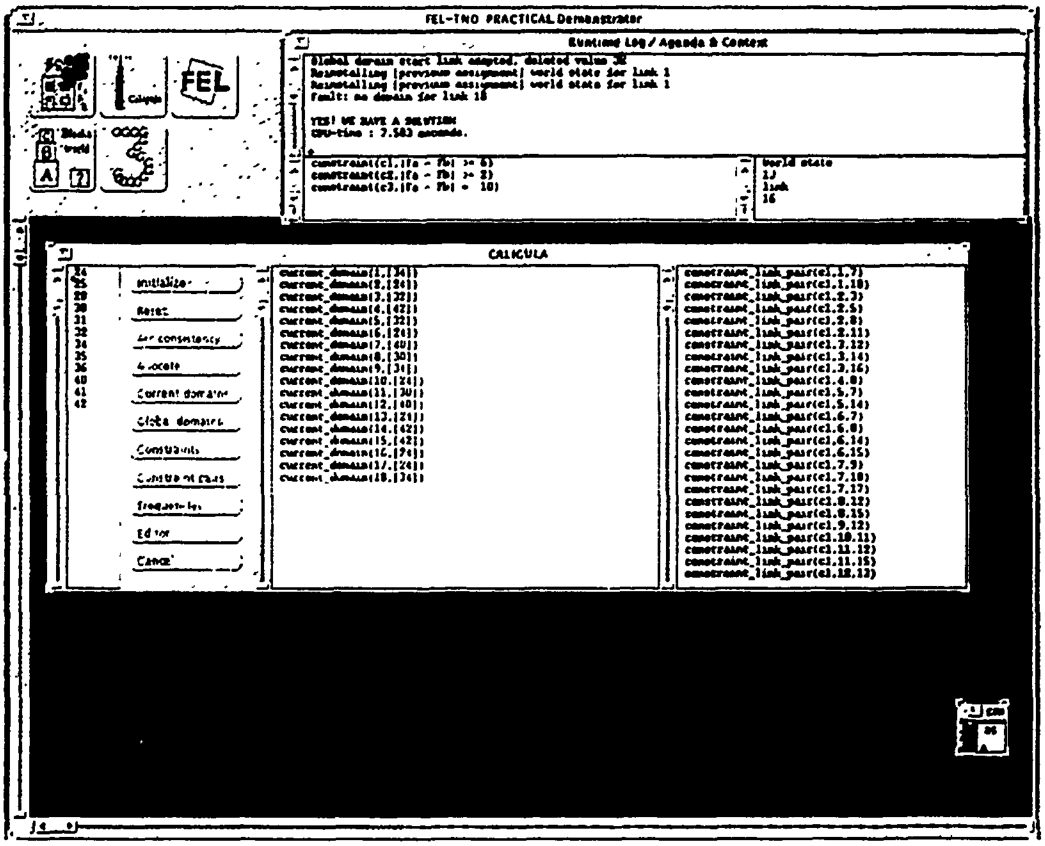
4 The madness of the Roman emperor here applied as a metaphor for the stupifying complexity of the combinatorial problem. Caligula's main feat during his reign was appointing his favourite horse as a colonel in the Roman army.

Hereafter, a global arc consistency run can be applied to the complete node network to propagate the effects of these overall inconsistent values through the net, hereby possibly decreasing the global domains of other nodes, possibly all. This is important, because for a next propagation run these global domains may be taken as the initial global domains. In this way, additional assignments of values to nodes will require less calculation and thus be acquired more rapidly.

We have used heuristics for determining the next best node for value assignment and for determining the choice of value from the current domains. Also, the context reinstallment mechanism is non-trivial. We will however not go into further details here. Figure 5.2 below is a screen dump of the CALIGULA demonstrator.

We have experimentally demonstrated the more than considerable gain in efficiency of constraint propagation. For the simple frequency allocation example that we have implemented a solution was found in 7.5 CPU seconds (SUN SparcStation10 workstation), whereas the generate-and-test approach (finding all solutions) which we have implemented as well would have required some 1000 CPU years (!). For a large (but realistic) frequency allocation problem with 1000 radio links, 100 frequencies and 2000 constraints on edges there would be 20 million calculations ($2000 * 100^2$) in worst case. On the basis of these results we estimate that a solution will be found in several CPU hours [5].

5 We note that the target set in the Project Specification of EUCLID RTP 6.4 is to find a feasible frequency assignment for a 100 values/1000 variables problem within two hours on a high-power Unix workstation.



```
Global domain start link adapted, deleted value 38
Reinstalling [previous assignment] world state for link 1
Reinstalling [previous assignment] world state for link 1
Fault: no domain for link 18

YES! WE HAVE A SOLUTION
CPU-time : 7.583 seconds.
```

```
constraint(c1,|fa - fb| >= 6)
constraint(c2,|fa - fb| >= 2)
constraint(c3,|fa - fb| = 10)
```

```
world state
13
link
16
```

```
24
25
28
30
31
32
34
35
36
40
41
42
```

```
current_domain(1,[34])
current_domain(2,[24])
current_domain(3,[32])
current_domain(4,[42])
current_domain(5,[32])
current_domain(6,[24])
current_domain(7,[40])
current_domain(8,[30])
current_domain(9,[34])
current_domain(10,[24])
current_domain(11,[30])
current_domain(12,[40])
current_domain(13,[24])
current_domain(14,[42])
current_domain(15,[42])
current_domain(16,[24])
current_domain(17,[24])
current_domain(18,[34])
```

```
constraint_link_pair(c1,1,7)
constraint_link_pair(c1,1,18)
constraint_link_pair(c1,2,3)
constraint_link_pair(c1,2,5)
constraint_link_pair(c1,2,8)
constraint_link_pair(c1,2,11)
constraint_link_pair(c1,3,12)
constraint_link_pair(c1,3,14)
constraint_link_pair(c1,3,16)
constraint_link_pair(c1,4,8)
constraint_link_pair(c1,5,7)
constraint_link_pair(c1,5,14)
constraint_link_pair(c1,6,7)
constraint_link_pair(c1,6,8)
constraint_link_pair(c1,6,14)
constraint_link_pair(c1,6,15)
constraint_link_pair(c1,7,9)
constraint_link_pair(c1,7,18)
constraint_link_pair(c1,7,17)
constraint_link_pair(c1,8,12)
constraint_link_pair(c1,8,15)
constraint_link_pair(c1,9,12)
constraint_link_pair(c1,10,11)
constraint_link_pair(c1,11,12)
constraint_link_pair(c1,11,15)
constraint_link_pair(c1,12,13)
```

Figure 5.2: Screen dump of CALIGULA

5.2 Scheduling

While static resource allocation deals with the allocation of resources to agents under several constraints neglecting the notion of time, in most real-life allocation problems however, time usually plays the role of the most restricting constraint. Moreover constraints on time can behave as meta-constraints on the applicability of 'normal' constraints. In the next four paragraphs we will address the scheduling problem as a natural but not completely trivial extension of the static resource allocation problem.

5.2.1 General formulation scheduling problem

The problem of scheduling can be considered a special case of the planning problem. Essentially, one has a set of activities to carry out while satisfying some set of constraints. Unlike the aforementioned planning problem however, these activities are usually completely detailed and do not have to be determined as in the planning problem (means-ends analysis). The question is how to combine these activities while maintaining whatever constraints are imposed by the problem domain. These constraints usually concern the availability of resources (such as a machine being available to perform one activity at a time) and temporal requirements (such as: the duration of a machine's availability before it is needed for another activity or a time frame in which all activities should be completed) [Georgeff87]. In addition, one usually wants an optimal or nearly optimal solution. Another observation one could make is that many of the constraints involved in scheduling are soft, although it is desirable that they be met they can be relaxed if necessary.

There are various standard mathematical techniques for solving scheduling problems. However, the combinatorial problems involved (see CALIGULA, paragraph 5.1.4) makes them unsuitable for most real-world applications, especially when rescheduling is involved. On the other hand there have been several attempts in AI to tackle this challenge.

5.2.2 Constraints

Scheduling - or as it is sometimes referred to: limited-resource time tabling - is often complicated since the constraints on the problem change so frequently that an algorithm for performing corrections to already allocated or scheduled resources is a necessity. Before outlining approaches that address this problem we first distinguish between several constraint types [d'Huy92a, Currie91].

- *Constraints on resources.*
  In which the following sub types can be identified:
  - Consumable resources
    These are resources that can be used only once, e.g. fuel for a car is a consumable resource.
  - Sharable resources
    These resources are not consumed, but can only be used by one agent at a time, e.g. a printer.
  - Renewable resources
    These are consumable resources that can create new resources, e.g. a piece of steel can be viewed renewable when it is used to create a container. The steel itself is no longer a resource but instead one has obtained a new resource.
- *Constraints on time.*
  For instance: deadlines, duration of actions, start time of actions, etc.
- *Constraints on the domain.*
  These constraints restrict the set of states of the world in which the scheduler is operating. For instance: a common constraint on military air bases is to distribute the aircraft over the base as much as possible in order to avoid clusters of aircraft which can lead to an unallowable vulnerability.
- *Context dependent constraints.*
  A special type of constraint is a context dependent constraint. In fact it isn't really a constraint type but a specialisation of the constraint types described above. The applicability of a context dependent constraint relies on the state of the world. For instance:
  - Constraints on incrementally changing resources.
    If a bank account has a constraint that one isn't allowed to have a deficit then this has an influence on the number of possible orderings of deposits and withdrawals from this account.
  - Constraints depending on the order of actions.
    If for two actions: *clean the floor* and *walk to the door* it holds that after cleaning the floor one is not allowed to walk over it for about five minutes then this constraint influences the start time of the latter action. On the other hand, if the

two actions were ordered vice versa they could be scheduled directly following each other.

5.2.3 Solving the scheduling problem

As the constraint types described above frequently occur in real-world scheduling problems it should be clear that these can't be solved by pure constraint propagation. The applicability of constraint propagation mainly follows from the ability to restrict a domain of values using the structure of the constraint. In order to perform restricting operations on the domains it is required that the constraints can be represented in a more or less mathematical way, which can't be done for every constraint type.

One can however try to follow a more intelligent approach in solving these problems. Many organisations use experts to solve scheduling problems. Generally, experts have gained much experience in and intuition on solving their specific problem and are therefore capable of finding an acceptable solution quickly. Beside the more or less mathematical approach followed by constraint propagation we would like to use (expert) knowledge in solving the more advanced scheduling problems.

We have used a rule-based approach [Gudes91] as a general paradigm for solving time-tabling problems. This paradigm includes generic concepts for resources, activities, constraints and allocations. The general control strategy is suitable for a large family of time-tabling problems and includes parts that deal with allocation, constraint checking and changes to allocations.

In [Gudes91] a framework for time-tabling is defined with the following three components:

- *Common set of concepts and terms.*
  Activities, i.e. a list of tasks to perform;
  Resources, i.e. a list from which allocations must be made to perform the tasks;
  Priorities. Activities are assigned priorities, which direct the allocation procedure;
  Allocations, the assignment of resources to activities;
  Constraints, i.e. restrictions on possible assignments that limit the overall possibilities. these can be either local (applicable to every allocation) or global (applies to the overall schedule).
- *Set of rules (tactics).*

Three classes are identified:
Priority rules, i.e. rules that assign priority amongst activities;
Restricting rules, i.e. rules expressing constraints;
Recommending rules, i.e. rules that recommend a specific allocation.

- *Strategy*.
  The strategy consists of three components:
  - Forward allocation using priorities and recommending rules;
  - Consistency checking. Does the current allocation meet the local constraints? The global constraints are checked at the of the process; [6]
  - Backtracking strategy. The standard depth-first policy is replaced by a strategy in which local change (minor swapping of allocations in the current schedule such that they do meet the constraints) and constraint relaxation are preferred above normal non-informed backtracking. Of course the relaxation of constraints is typically controlled by meta-rules.

5.2.4 ALLOCATOR

Following the article by Meisels, Kuflik and Gudes [Gudes91], we have implemented their framework in Quintus Prolog [Prolog], and used it to build a demonstrator in which crew assignments to airborne missions have to be made (called ALLOCATOR). The assignments result in a schedule of allocations for one week. The next goal of the demonstrator was to prove the capabilities of the framework in supporting dynamic re-allocation. In other words, the schedule should be maintained dynamically as time passes and missions are flown. It should be clear that for each mission the duration is estimated, during the flight this estimation is adjusted which may affect the schedule of next missions to be flown. Moreover, resources may no longer be available due to for instance sickness of pilots, unexpected maintenance, accidents or even crashes.

To solve the assignment of airborne missions, the program has to create a weekly assignment of crews to their activities. Each crew is composed of three crew members, who may have three different types of rank. Crew members of higher rank ca serve in lower-ranked roles on a mission, but not the other way around. Crew members of all types require a special qualification for each

6 This as opposed to constraint propagation.

task to be performed on a mission. There are about 16 activities of different kinds to be performed each week. Every kind of activity has a separate set of (±48) tasks to perform on it. Crew members are selected from a pool of about 27 crew members, where everyone has some characteristic availability, type (rank), qualifications for tasks, and restrictions on these qualifications.

In accordance with the framework, the following entities and rules were defined:

Activities: Missions to be performed.

```
Mission:
Name            : Name of the mission (unique).
Type            : {fi, ah, ch, op, ge}.[7]
Day             : {su, mo, ..., sa}.
Start time      : 0h00 ... 23h59.
End time        : 0h00 ... 23h59.
Role            : Each mission has three roles: c, r, a.
```

Resources : Crew members.

```
Crew member:
Name            : A unique identifier.
Accessibility   : Regular, career or on reserve: sl, re, sr, rr.
Accredition     : Three hierarchical types: c, r, a.
Availability    : List of days when member can be appointed.
fi-counter      : The number of fi missions that have been assigned.
ah-counter      : The number of ah missions that have been assigned.
ch-counter      : The number of ch missions that have been assigned.
op-counter      : The number of op missions that have been assigned.
ge-counter      : The number of ge missions that have been assigned.
```

Recommending rules (only a sample is given):

```
1. Select mission of type 'fi'.
   Assign to the role 'c' a staff-man such that:
   - his/her accredition equals 'c',
   - his/her accessibility equals 're' or 'sr',
   - the day of the mission is one of the days in the availability list,
   - the total number of missions of type 'fi' is larger than the average.

2. Select a mission of type 'fi'.
   Assign to the role 'r' a staff-man such that:
```

7 This is specific Air Force terminology

- his/her accreditation equals 'c',
- his/her accessibility equals 're' or 'sr',
- the day of the mission is one of the days in the availability list,
- he/she was already scheduled to fly on this day,
- the total number of missions of type 'fi' is smaller than the average.

Restricting rules:

1. A crew member cannot be assigned to two activities that overlap in time.

2. The same crew member cannot be assigned twice (to two different roles) on the same activity.

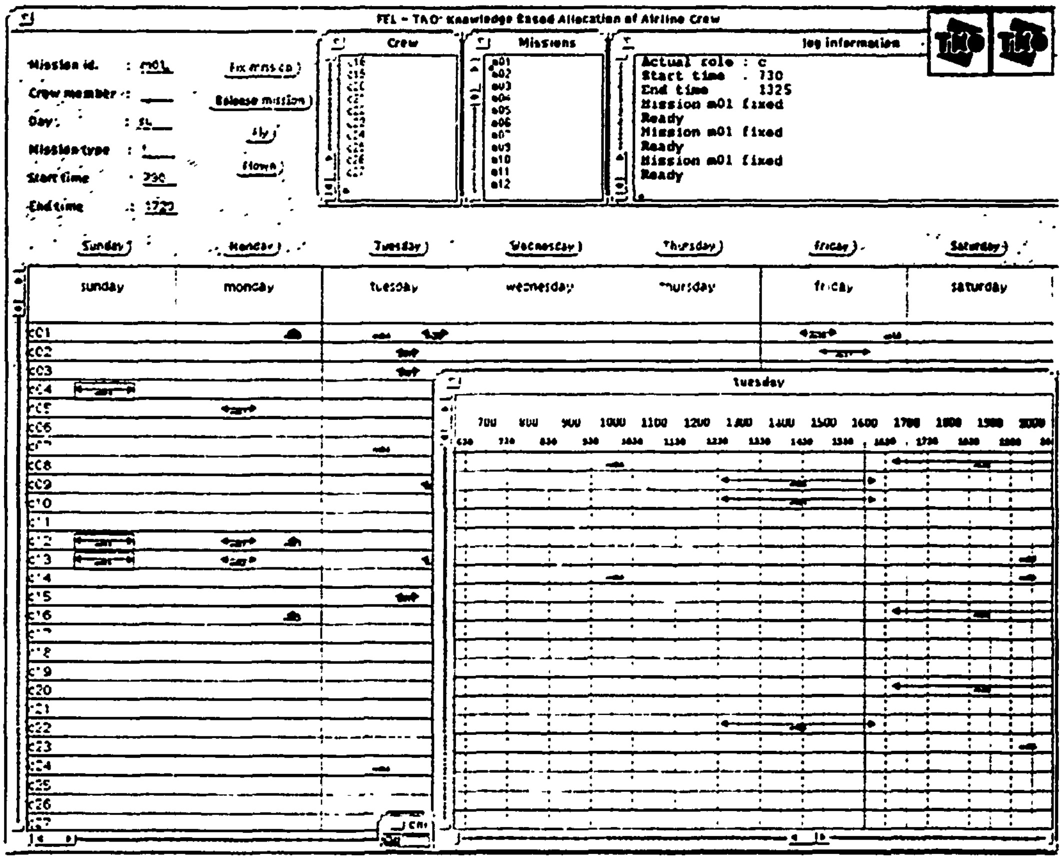
3. A crew member must have at least a period of 30 minutes between two assignments.

Priority rules:
1. Assign highest priority to missions of type 'fi' then to 'ah' then to 'ch'.

In figure 5.3 below a screen dump of ALLOCATOR is presented. On top of the main window one can see an enlarged view for the allocations for flights scheduled on Tuesday. Using mouse & keyboard, the user can manipulate the schedule in several ways. For instance one can:

- add or delete crew members;
- add or delete missions;
- change the attributes of the crew members and missions (types, qualifications, etc.);
- change the attributes of allocations (actual start time, actual duration, etc.).

Moreover the system dynamically calculates the effects of these alterations for the current schedule. Allocations can be fixed in order to be able to express that the corresponding missions have been flown or are currently airborne.



Mission id. : m01
Crew member :
Day : su
Mission type : f
Start time : 730
End time : 1720

```
Actual role : c
Start time  . 730
End time      1325
Mission m01 fixed
Ready
Mission m01 fixed
Ready
Mission m01 fixed
Ready
```

Figure 5.3: Screen dump of the ALLOCATOR system.

5.3 Dynamic resource allocation in (re)planning

Apart from static resource allocation and scheduling we distinguish dynamic resource allocation as it is used in a planning system. The main task of a planning system is to determine a course of actions which when executed achieve a desired state of the world. These actions use resources, therefore a subtask of a planning system is to allocate resources to these actions.

An important difference between allocation in scheduling problems and allocation in planning problems is that in the former case one is working with a predetermined set of actions while in the latter case the set of actions has yet to be determined. The main consequences of this difference are:

- In scheduling the domain of values for allocations is more or less fixed as it only changes due to allocations of the scheduler. In planning however, this domain is dynamic (especially in C^2-planning) and each time the planner needs an allocation the domain to choose from will first have to be determined.
- If an attempt to find a resource for an action fails then scheduling has three options to continue with:
    - *Local change*, i.e. swapping a resource with a previous allocation.
    - *Constraint relaxation*, i.e. assigning a resource that normally would violate a constraint but which is applicable under very strict conditions (more rigid constraints on other allocations).
    - *Backtracking on allocations.*

In planning however it is possible that beside these previously mentioned options another option is available. One could try to use another tactic (course of actions) which may lead to satisfiable allocations.

On the other hand one of the main requirements for a tactic to succeed is the availability of resources to implement it. In other words, when incorporating a tactic in a plan, one has to know whether it is implementable or not. Therefore allocation of resources to actions directly follows the incorporation of the actions into the plan. This as opposed to scheduling in which an allocation is done over the complete set of actions.

When an action is going to be asserted to the plan the planner first allocates all the necessary resources to perform it. Hence, it has to dynamically determine the current domain of assignable resources. This is done by retrieving all unassigned resources that are available during a specified time window[8]. Typically this window has a length equal to the estimated time for an action. Sometimes the estimated time for achieving the goal is used, for instance when several actions within the script that achieves the goal use the same resource, see figure 5.4 below.

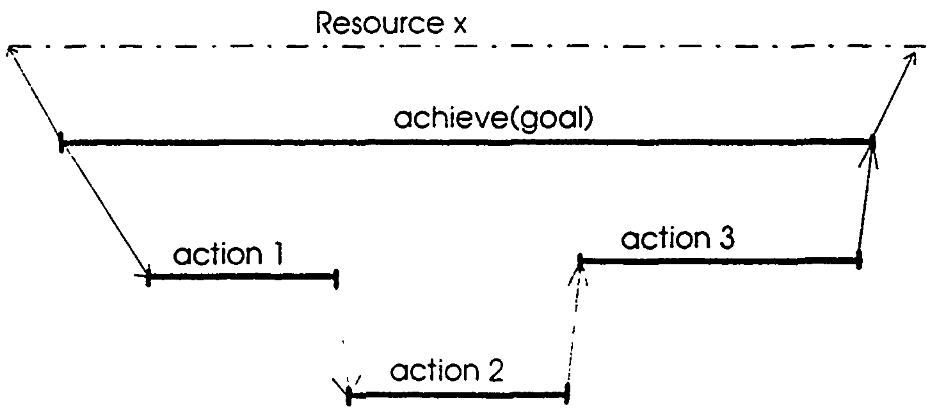


Figure 5.4: The claiming of a resource during a goal

When an allocated resource isn't available during execution of the plan, re-allocation can be done in the same way as described above. When there aren't any free resources left a local change or a constraint relaxation may be possible. If these three all fail a different tactic can be tried as described in section 4.2.

8 This can be done using the 'occupied' query as described in paragraph 4 1.3.

6 The PRACTICAL demonstrator

In this chapter we will describe the functionality of the PRACTICAL demonstrator and assess its merits. We will address the various functions of the demonstrator by describing the menu options. The implicit solutions pertaining to (re)planning as discussed in the previous chapters will be discussed as well as performance issues. In the concluding remarks of the next chapter we will discuss limitations and the applicability to command and control problem domains.

6.1 Outline of the demonstrator

The main menu of PRACTICAL first of all contains a choice of application. An application included is the fire contingency used as an example in chapter 4. Furthermore, the demonstrators CALIGULA and ALLOCATOR are included and can be run independently.

The options pertaining to event handling are the following:

- `Event`
  ```
  New event
  Feedback event
  ```
- `Handle events`
  ```
  First Come First Served
  Conflict Resolution
  ```

Selecting *New event* will enable a popup window in which a new world event can be entered. For the fire contingency application we have included a database of possible events that can be selected by mouse-clicking. The choice *Feedback event* is a function allowing world events to be entered. This will typically take place during plan execution to assert altered distances of executed action occurrences, i.e. that are not equal to the distances as predicted in the plan. Also, new fact occurrences, action occurrences inserted manually in plans, protections, possibly clips or even new goals can be asserted to the temporal database in this way, along with corresponding distances. To this end both an *occurrence editor* and a *distance editor* have been incorporated. The functionality of these editors is more extensive, this will be illustrated during the course of this section.

When a (series of) event(s) have been entered, the next action is to let the planner handle these events. This implies that the event rule base is called upon by PRACTICAL to connect the event(s) via the scenario/script indicators to applicable scripts. An included option is the strategy applied to this (forward-chaining) rule base: first come first served or applying conflict resolution when more than one event rule is applicable. The result is that the script headers of the thus chosen scripts are the top goals to be achieved by the planning system, these are inserted on the goal agenda. Optionally the goal selection rules may be applied here as well to determine an initial order for the goal agenda. This implies that the order of the goal agenda is not necessarily determined by the order in which applicable scripts were selected, nor necessarily chronological with respect to the predicted starting times of the goals on the agenda.

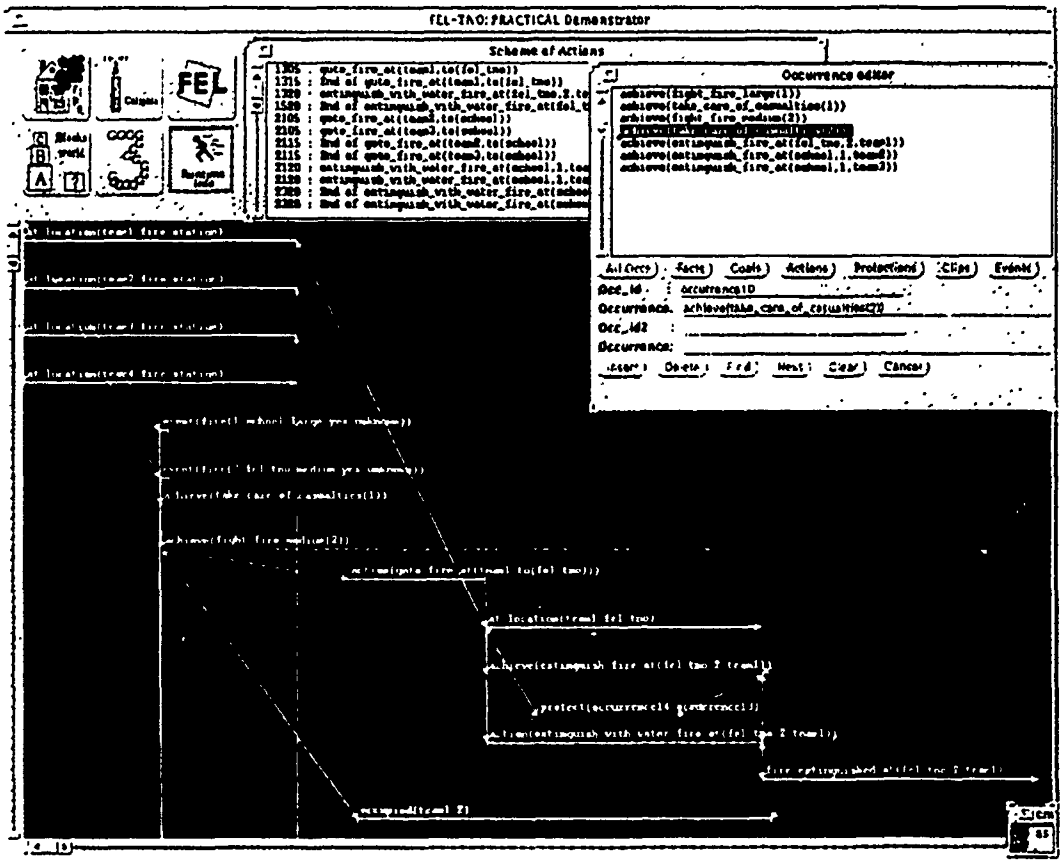
When the goal agenda has been installed the plan can be calculated. The options for planning are twofold, the default option is to plan all the goals on the agenda, but it is possible to plan a single goal, further enhancing a "piecemeal" approach to planning:

- Plan
```
Plan all
Plan goal
```

The results of the plan can be consulted in various ways. We have implemented a drawing facility allowing to view the complete plan with the following options:

- Draw
```
Draw logic timemap
Draw real timemap
Draw goal
Clear timemap
Canvas selection
To file
```

The complete timemap that results from the planning process can be viewed on a canvas. The first option is viewing the timemap by the relative orderings amongst all occurrences (logic timemap). It is also possible to view the timemap with respect to absolute time, thus with aligned start and end timepoints and durations of occurrences by ratio (real timemap). A single goal can be viewed with its expansion. Furthermore, the timemap can be cleared, the size of the canvas can be adjusted and the timemap can be dumped in a file. To illustrate the drawing facility, figure 6.1 is a screen dump of (a portion of) a resulting timemap of a fire contingency plan for two fire events.



```
1305 : goto_fire_at(team1,to(fel_tno))
1315 : End of goto_fire_at(team1,to(fel_tno))
1320 : extinguish_with_water_fire_at(fel_tno,2,te
1520 : End of extinguish_with_water_fire_at(fel_t
2105 : goto_fire_at(team2,to(school))
2105 : goto_fire_at(team3,to(school))
2115 : End of goto_fire_at(team2,to(school))
2115 : End of goto_fire_at(team3,to(school))
2120 : extinguish_with_water_fire_at(school,1,tea
2120 : extinguish_with_water_fire_at(school,1,tea
2300 : End of extinguish_with_water_fire_at(schoo
2300 : End of extinguish_with_water_fire_at(schoo
```

```
achieve(fight_fire_large(1))
achieve(take_care_of_casualties(1))
achieve(fight_fire_medium(2))
[illegible]
achieve(extinguish_fire_at(fel_tno,2,team1))
achieve(extinguish_fire_at(school,1,team2))
achieve(extinguish_fire_at(school,1,team3))
```

Occ_Id : occurrence10
Occurrence: achieve(take_care_of_casualties(2))
Occ_Id2 :
Occurrence:

Figure 6.1: Screen dump of PRACTICAL for two fire events

Apart from the occurrence and distance editors, allowing the manipulation of the temporal database, the resulting scheme of actions - essentially the plan to be executed - can be viewed. An example of this can also be found in figure 6.1.

For the temporal path distance calculations carried out in the planning process the Ford algorithm (see chapter 3) is used by default. The Practical Path algorithm implemented can be optionally used as well. The agenda can be consulted, the contexts can be selected independently, allowing the viewing of the :volution of the plan along with the drawing facility. Thus, the complete plan history is kept and can be consulted. Of course, the previous contexts are used mainly for replanning.

-
  ```
  Replan
          Replan all
          Replan action
          Replan goal
  ```

Replanning, described in chapter 4, is an interactive process in PRACTICAL. The operator of the system indicates which action(s) and which goal(s) must be replanned. This is done by using the occurrence and distance editors. For instance, when an action that has been executed results in a duration different than predicted by the plan this (feedback event) is asserted in the temporal database by adapting the appropriate distances. Selecting the action in the occurrence editor and clicking the menu option for the replanning of the action will result in the goal hierarchy being checked for the parent goal of the action and possibly their parent goals, as described in chapter 4. Results of the replanning concerning adaptations in the bounds (durations) of the respective goals, possible resulting conflicts concerning resources and dependent follow-up goals will be presented to the operator. A current omission in PRACTICAL are explicit suggestions for plan repair due to these conflicts, this is up to the operator at this point, thus further extensions of interactive replanning with additional knowledge pertaining to plan repair are necessary.

The knowledge partitioning used as the basis for the PRACTICAL framework is explicitly contained in the menu and can be viewed. The options are the following:

-
  ```
  Knowledge
  ```

    -
      ```
      Strategic
         Script selection rules
            FCFS
            FCFS alternative
            Intelliguess
         Goal selection rules
            FCFS
            Earliest goal first
            Intelliguess
      ```

    -
      ```
      Tactical
            Scripts
            Constraints
      ```

    -
      ```
      Causal
            Adders
            Clippers
      ```

```
•    World
           All occurrences
           Facts
           Actions
           Goals
           Protections
           Clips
           Distances
```

Script and goal selection as have been discussed can be applied manually. The "first come first served" (*FCFS*) option will select the first applicable script or goal respectively. The *FCFS alternative* is used by PRACTICAL for tactical options in replanning; the scripts applied are kept track of, thus when FCFS finds a script already used to expand the culprit goal being replanned it will choose the alternative if available. The *Earliest first* option for goal selection will choose the chronologically earliest goal (with respect to the start timepoint), the *Intelliguess* options for both script and goal selection explicitly use the strategic (meta)knowledge of the script and goal selection rules.

The scripts and constraints (preconditions, resource allocation constraints in scripts) can be viewed in windows. The adders and clippers can be consulted as well. The world knowledge can be viewed with the occurrence and distance editors; selecting for instance *Facts* will present all the facts in the browser of the occurrence editor. These options are also selectable in the editors themselves. The distance editor has additional options for the temporal queries available in PRACTICAL, for instance determining whether an occurrence is "true-throughout" an interval determined by two timepoints, or an overlap query pertaining to two occurrences. Figure 6.1 contains also an occurrence editor.

6.2 Implicit solutions

The above (compact) description illustrates the approach to planning taken in PRACTICAL. Here we will sum up the implicit solutions already addressed in the previous chapters and address some performance issues.

The framework of the PRACTICAL system implies first of all a strict partitioning of knowledge used for planning and replanning. The mapping of this framework onto the demonstration system was achieved by the implementation of this knowledge in terms of world events, scenario indicators, script and goal selection rules, the scripts themselves and constraints on these, the adders and clippers and the world knowledge expressed in occurrences and distances in a temporal database. A major advantage of this explicit partitioning is that the knowledge is *manageable*. The rule bases are easily adapted. Scripts can be adjusted easily and new scripts can be added. Adder and clipper rules are separately contained in a rule base. Occurrences and distances can be manipulated independently.

The temporal database management facility has the advantage that reasoning about a *relatively ordered* time frame is possible. This is not so if time-stamping is applied. We have taken a timepoint-based approach to this temporal context, thus adopting the point as the basic entity of time [**McDermott82**]. The alternative is an interval-based approach, thus taking the basic temporal entity to be an interval, implying however that one can only reason about temporal events in terms of coming before or after another event, or overlapping with another event [**Allen91**]. With the timepoint-based approach one is able to speak of absolute timepoints as well as intervals, the latter constructed from two timepoints. This approach is justifiable because first of all this is a more intuitively natural approach to time, but moreover from a computational viewpoint because the interval approach is computationally intractable, algorithms will run in non-polynomial time [**Kautz86**]. This as opposed to using Ford for a timepoint-based approach, in chapter 3 it has been clarified that the computational complexity is then of $O(en)$, the product of the number of edges and the number of nodes (vertices). The number of edges is in worst case proportional to the quadratic number of nodes. Therefore the timepoint-based approach using Ford's algorithm implies tractability, namely proportional in worst case to $O(n^3)$ with n being the number of nodes. In practice this will imply a performance somewhere between quadratic and cubic time.

The table in figure 6.2 shows the results of the planning effort necessary for the fire contingency problem for varying numbers of fire events and allocable resources (fire teams in this case). A single fire event is planned in several CPU seconds. With 8 events and 15 resources the plan is computed in some 4.5 minutes. The fire contingency problem implemented is admittedly simple, however, we believe the results to be promising, an extensive plan that is calculated in say half an hour by a computer is a far better result than a comparable effort of hours or even days when applied manually. The predicted performance is evident here, between quadratic and cubic time with the number of events as a proportional indication of the number of nodes involved.

# Events	5 resources	15 resources	25 resources
1	2.3 sec.	4.5 sec.	6.7 sec.
2	6.0 sec.	15.8 sec.	n.a.
3	19.2 sec.	26.2 sec	n.a.
4	n.a.	52.9 sec.	66.5 sec.
8	n.a.	277.8 sec = 4.5 min.	n.a.

Figure 6.2: Performance results of fire contingency example

The frame and ramification problems have been addressed (see chapter 4). However, this does imply an effort concerning acquisition and compilation, in particular for the *adders*. These will have to incorporate frame implications (direct consequences) and ramification implications (indirect consequences, travelling via implication rules). The set of adders will have to be *complete* in the sense that all implications will have to be represented to ensure completeness of ramification.

7 Concluding remarks

The framework and the PRACTICAL demonstrator described in this report provide a scenario-based approach to planning and replanning with (partial) re-use of the originally developed plan. The framework is generic, it does not depend on the domain at hand in which a plan is developed. This enhances a flexible approach to planning, allowing for generic scripts that are selected in response to an event or series of events taking place in the world. It furthermore allows the analysis of strategic options, implying possibilities for "what-if" analyses.

The planning system described in this report uses techniques from the field of Artificial Intelligence. It is not our intention however to accentuate the notion of "artificial" in this term. The aim of every automated system should be to support the user, in our case a planner in a command and control domain, as best as possible, allowing for an interaction and leaving the user in full control. The dynamic replanning facilities incorporated in PRACTICAL are based on precisely this requirement. We would therefore rather speak of information systems with intelligent support.

An automated system always raises questions concerning the performance bottleneck. As discussed in chapter 6, the PRACTICAL system does well on this count. This is due to the fact that we use algorithms with complexities in the order of the number of edges times the number of nodes for a (virtual) tree spanning the problem domain. Therefore, the potential exponential growth when planning in ever larger domains does not imply an equivalent computational growth when solving the planning problem. Memory requirements for the domain and for the context saving mechanism pose no problems for state-of-the-art computers.

Another potential bottleneck for a knowledge-based system pertains to knowledge acquisition. In particular, the scripts entail a considerable compilation effort necessary prior to being able to benefit from a scenario-driven approach to planning. However, this is true in general because scenarios and scripts describing e.g. red-white-blue force command and control options will necessarily have to be available for manually planned missions and operations as well in the form of tactical doctrine. An additional effort is thus necessary to implement the scripts in the formal notation required in the PRACTICAL framework, incorporating preconditions, resource allocations and temporal database instantiations.

Further research is necessary, particularly focusing on interactive replanning. The replanning facilities implemented in PRACTICAL are promising in terms of tractability and efficiency and means for partial re-use of existing plans (see chapter 6). Further expansions of the (re)planning algorithms will have to be investigated in terms of heuristics and other options for increasing efficiency. However, a generic interactive replanning component has yet to be implemented with knowledge-based support for the operator. This will incorporate facilities partly available in PRACTICAL for replanning during plan execution such as means for re-allocation of resources in case of conflicts, suggestions for expanding temporal bounds for goals that are not fulfilled in the predicted durations and interactive conflict resolution for follow-up activities that are effected by these changes.

PRACTICAL is built in Prolog and uses the Prolog predicate database, extended with a temporal database management system (TDBMS) that is also Prolog-based. An operational planning system should have enhanced database management facilities (especially pertaining to backup and recovery) and use commercial-off-the-shelf (COTS) software as much as possible. However, these would have to incorporate TDBMS facilities. To this end there are several options. The TDBMS could be realised in a deductive database system (e.g. the ADITI system, however currently a Beta-version [ADITI]). Another option is using POSTGRES, the INGRES RDBMS follow-up with among others facilities for spatial and temporal database management (albeit time-stamping-based) [POSTGRES, Stonebraker90], becoming commercially available this year. Also, the use of the RDBMS ORACLE with the (existing) bridge to Prolog is an option worth considering. Finally, a next generation object-oriented DBMS (for a survey of OODBMS's, see [Everest92]) such as ONTOS [ONTOS] could be used, however necessitating the additional coding of all temporal database management facilities (for ONTOS in C++). There are several commercial packages dedicated to resource allocation and scheduling (however not planning in the sense as described in this report) such as CHIP [CHIP] and SCHEME [SCHEME]. These have the disadvantages that they were not designed for Command and Control domains, do not provide a generic framework and are based mainly on numerical algorithms and operations research techniques such as dynamic programming, implying a lack of a knowledge-based foundation.

Planning systems such as the one described in this report will potentially benefit from other current research projects, and vice versa. In particular, there are several EUCLID (European Co-operation for the Long term In Defence) projects that will start in the near future: RTP 6.1 (Command and Control workstation, accentuating (re)planning), RTP 6.4 (Combinatorial algorithms for resource allocation problems) and RTP 6.5 (Crew Assistant on-board single seat fixed wing aircraft). We also refer to the Army Tactical Command and Control Information System (ATCCIS) incorporating a planning module currently being developed.

The applicability of scheduling and planning systems to C^2-domains is extensive. In a military environment possibilities are air force mission scheduling and both pre- and inflight mission and tactical planning, operations planning for G3-sections at army brigade/division level, damage control on board naval ships and naval operations planning. In a civilian environment applications are for instance in the area of fire brigade support, contingency planning in general, police operations, Coast Guard operations and chemical processing industries. This broad scale of application implies a necessity of accentuating the generic character of frameworks and tools for scheduling and (re)planning. Current research at FEL-TNO in the context of planning in the areas described above is directed at naval damage control (the extension of the prototype system DAMOCLES to the operational on-board system ANDES), the participation in the ATCCIS project, the further extension of the PRACTICAL framework, building an application for air force mission scheduling for operational use in the Squadron Operations Centre (SQOC) and the extension of such a scheduling tool to a generic level (as in PRACTICAL) to be able to incorporate goals, activities, resources, constraints and temporal management facilities for the complete range of C^2-domains.

[illegible]

C.W. d'Huy
(author)

[illegible]

A.P. Keene
(author)

[illegible]

M.R. Woestenburg
(project leader)

Abbreviations and acronyms

AI	Artificial Intelligence
ALLOCATOR	Allocation Tool for Operations
ANDES	Advanced Naval Damage control Expert System
ATCCIS	Army Tactical Command and Control Information System
ATMS	Assumption-based Truth Maintenance System
C^2	Command and Control
CALIGULA	Combinatorial Algorithms Guiding Radio Link Frequency Allocation
CCIS	Command and Control Information System
COTS	Commercial Off The Shelf
CPU	Central Processing Unit
CSP	Constraint Satisfaction Problem
DAMOCLES	Damage Monitoring and Control Expert System
DBMS	Database Management System
EUCLID	European Cooperation for the Long term In Defence
FEL	TNO Physics and Electronics Laboratory
G3	G3 Operations Staff Section
ISD	Information Structure Diagram
KBS	Knowledge-based System
MMI	Man Machine Interface
NIAM	Nijssens Information Analysis Method
NOAH	Nets of Action Hierarchies
OODBMS	Object-Oriented Database Management System
POSTGRES	Post INGRES
PRACTICAL	Planning and Resource Allocation in C2-domains with Time Critical Algorithms
PROLOG	Programming in Logic
RDBMS	Relational Database Management System
SQOC	Squadron Operations Centre
TDBMS	Temporal Database Management System
TMS	Truth Maintenance System
TNO	Netherlands Organization for Applied Scientific Research
YSA	Yourdon Structured Analysis

Glossary

Activity

An activity is the basic entity of the planning domain. Activities may be organised in a hierarchy, therefore we distinguish between primitive and composite activities [Vadon91].
Synonym: task,action.

Agent

Activities are performed by agents (humans, machines, robots). There may be one or more agents (multi-agent planning) that can perform one or more activities at a time.

Algorithm

A method for solving a combinatorial problem. Examples are breadth-first, depth-first, branch and bound, A*, hill-climbing, constraint propagation.

Allocation

The assignment of resources to activities. Assignments can be permanent or temporary [Gudes91].

Arc consistency

During constraint propagation the allocation of resources to activities must be such that the constraints are satisfied. This is accomplished by applying arc consistency. An arc is a pair of activities that is subject to some constraint. Each activity has a domain of possible resources. Arc consistency implies that for each resource in the domain of arc A there must be a resource in the domain of arc B such that the constraints between arc A and arc B are satisfied. If this is not the case the resource in the domain of arc A is eliminated. Arc consistency is directional: if arc(A,B) is consistent this does not imply that arc(B,A) is consistent as well.

Backtracking

A strategy for tracking back into the planning process in case a feasible solution cannot be found. The backtracking strategy will "cut" the non-feasible path and try a next one.

Combinatorial problem

In non-trivial problem domains the number of alternatives that have to be explored for a solution grows exponentially in terms of the number of nodes in the search space and the number of paths leading from the start node. This exponential growth is known as the combinatorial explosion.
See also: exhaustive search.

Complete ordering

All planning systems developed before 1975 used a complete ordering of actions in time. When a new action had to be added, the system knew exactly which actions came before it and which actions came after it. Since the introduction of Sacerdoti's NOAH **[Sacerdoti77]** the complete ordering of actions was weakened to a partial ordering. Complete ordering of actions is left until the time it is really needed. Working with a partial ordering gives the planner the flexibility to overcome unexpected interactions between actions.
See also: partial ordering.

Constraint

Planning and scheduling problems usually have constraints on the set of possible solutions. These constraints can be used to prune the search space and/or to direct the search for a solution. We distinguish several types **[Currie91]**:

- Constraints on resources;
- Constraints on the time;
- Constraints on the domain.

Each of these can be hard or soft constraints (must be met/ should be met if possible).
See also: constraint propagation, exhaustive search.

Constraint propagation

Instead of focusing on the resources and activities and checking constraints *after* allocation, constraint propagation focuses on the constraints for deriving a next step in the solving process **[Kumar92]**. An activity is chosen (according to some heuristic), the corresponding constraints are satisfied, resulting in a set of activities for which resources are allocated or for which the set of possible resources is restricted. Of these activities one is chosen, etcetera. Thus the constraints propagate through the sets of activities and resources, and are satisfied along the way.

Context
A context in PRACTICAL is the state of the plan evolution at a certain moment. It contains all occurrences (facts, actions, goals, protections, clips) and all distances as well as the goal agenda. A context is used for replanning to be able to retrieve a previous situation of the plan development.

Domain
A domain can be defined as the environment where the actions of the planner have an impact on. This environment defines the possible actions, resources and constraints that the planning system should work with.
See also: state, world.

Duration
Actions can be instantaneous or they can have a certain duration. Intuitively actions do have a duration. The duration can be unknown, known exactly or vary in a certain time interval. When actions can be executed concurrently they may overlap.

Event
We define an event as something that happens under control of the planning/scheduling system. Events have a bounded (often known) duration. Events change the state of the world by clipping or setting persistences.
See also: persistence.

Exhaustive search
An exhaustive search of the search space of possible solutions by a "brute force" generate-and-test algorithm tries each alternative sequentially, and is an extremely expensive way of solving a problem. Most problems cannot be solved by exhaustive search due to hardware limitations.

Expansion
An expansion is the refinement of a goal in terms of subgoals. These subgoals may be primitive activities, or composite activities, the latter will be expanded in a next step of the planning process.

Goal

A goal can be viewed as a state of the world to be achieved. A way to represent such a goal is a conjunction of facts which have to be true in the goal state of the world.

Heuristic

A rule of thumb. Heuristics are very useful for the representation of knowledge concerning expert experience or strategic knowledge, e.g. for deciding which should be the next activity for resource allocation.

Knowledge

Knowledge about a domain comprises facts about the domain, rules describing relations between domain facts, and methods and heuristics for solving problems in the domain. Rules, methods and heuristics can originate from written knowledge - procedures, laws, etcetera - but also from (objective) empirical knowledge, (subjective) intuitive knowledge, and heuristic knowledge (rules of thumb). Knowledge can be represented in facts, relations, frames, objects, functions, rules and scripts.

Partial ordering

An ordering where some of the activities may be unordered. From the mathematical point of view partial ordering (reflexive, transitive, anti-symmetric) is incorrect, it should be replaced by quasi-ordering (irreflexive, transitive, anti-symmetric) [**d'Huy 92a**].
See also: complete ordering.

Persistence

Contrary to an event, a persistence is a fact that becomes true (due to an event) and remains true until another event changes (clips) it.
See also: event.

Plan

A plan is a structured set of actions which when executed in the real world transform the world from an initial state to the goal state.
See also: planning, schedule.

Planning

The word planning is used very loosely by the AI community (and others) to cover a range of problem types. Historically, 'planning' has referred to activity planning as exemplified by the blocks-world problem. In activity planning the solution is usually in the form of a partially-ordered sequence of actions which transforms an initial state into a goal state [**Gadsden88**].
See also: activity, plan, state.

Priority

An assignment of relevance amongst for instance goals, constraints and activities.

Replanning

Even automatic planning or scheduling is a time consuming business, therefore when a plan has to be revised, for instance to meet new or changed goals, we want to re-use as much as possible of the "old" plan, instead of planning all over again.

Resource

We distinguish the following resources [**Currie91**]:

- Consumable resources, can be used only once;
- Sharable resources, can be used several times but by one agent at a time;
- Renewable resources, these are consumable resources which create other resources.

Rule

A formalism for knowledge representation stated in an "if .. then .." format. The interpretation may vary, e.g.: if precondition P then conclusion C; if situation S then action A; if conditions C1 and C2 hold then condition C does not hold.
See also: expansion.

Schedule

A schedule is a plan that contains resource usage information with respect to time.

Scheduling and resource allocation

The problem of scheduling can be considered a special case of the planning problem. Essentially, a set of activities has to be carried out while satisfying a set of constraints. Unlike the aforementioned, however, these activities are known. No analysis of achieving a goal world, leading to a set of actions, is required because all the activities are already given. The question is how to order these actions while maintaining whatever constraints are imposed by the problem domain. These constraints usually concern the availability of resources (resource allocation) and temporal requirements. A special category is resource allocation with a fixed sequence, here the sequence is fixed; the task have been prearranged. All that has to be done is to allocate resources to the tasks. The components of a planner that determine the actions satisfying a goal and the ordering of these actions cannot be completely separated because goal analysis necessarily performs partial ordering.

Search space
The total space spanned by all the alternatives that may be solutions to a specific problem. It is comprised of combinations of activities and resources and can be (mentally) depicted as a tree.

State
Both the planning system and the world in which we can perform actions derived from the planning system have states. Each time an action is executed in the world the state of the world changes. Each step in the planning process changes the state of the planning system. Both the world and the planning system have a history defined by an ordered set of states.
See also: domain, world.

Task
A task can be viewed as a synonym for action or activity, but sometimes it is used as a task for the planning system, i.e. the task is to achieve a certain state of the world. We prefer to use "goal".
See also: activity, goal.

Task planning
Task planning is aimed at determining a set of actions that, when executed in a certain order, will change the current state of the world into the world specified by a prescribed goal, violating a minimum number of constraints on the domain.

Time
Time can be viewed in two different ways. The first view of time is an ordered set of very small points (timepoints), e.g. [McDermott82]. As opposed to this view, the primary temporal entity can

be taken to be an interval, e.g. [Allen83]. In the PRACTICAL framework the first view has been embraced.

World

The external domain on which the planning/scheduling activities have an impact. A planning system can have an interface to the real world or have a model of the real world. Ideally the model of the world has to correspond to the real world at every state.

See also: domain, state.

References and further reading

[ADITI] ADITI: A deductive database system, University of Melbourne, Parkville, Victoria, Australia.

[Aho75] A.V. Aho, J.E. Hopcroft, J.D. Ullman (1975), *The design and analysis of computer algorithms*, Addison Wesley, Reading, MA.

[Allen83] J.F. Allen (1983), "Maintaining knowledge about temporal intervals", in: *Communications of the ACM* **11**, pp. 832-843.

[Allen91] J.F. Allen, H.A. Kautz, R.N. Pelavin, J.D. Tenenberg (1991), *Reasoning about plans*, Morgan Kaufmann Publishers, San Mateo, CA.

[Andriole86] S.J. Andriole, H.H. Black, G.W. Hopple, J.R. Thompson (1986), "Intelligent aids for tactical planning", in: *IEEE Transactions on Systems, Man and Cybernetics* **16(6)**, pp. 854-864.

[Bonder86] S. Bonder, R.L. Farrell, G. Miller, L.D. Proegler, D.E. Thompson (1986), "Capturing Expertise: Some approaches to modelling command decision making in combat analysis", in: *IEEE Transactions on Systems, Man and Cybernetics* **16(6)**, pp. 766-773.

[Bratko86] I. Bratko (1986), *Programming for Artificial Intelligence*, Addison-Wesley Publishers Inc., Reading, MA.

[Bruynooghe81] M. Bruynooghe (1981), "Solving combinatorial search problems by intelligent backtracking", in: *Information Processing Letters* **12(1)**, pp. 36-39.

[Bruynooghe84] M. Bruynooghe, L.M. Pereira (1984), "Deduction revision by intelligent backtracking", in: J.A. Campbell (ed.), *Implementations of Prolog*, Ellis Horwood, pp. 194-215.

[Charniak87] E. Charniak, C.K. Riesbeck, D.V. McDermott, J.R. Meehan (1987), *Artificial Intelligence Programming*, Lawrence Erlbaum Associates, Hillsdale, NJ.

[CHIP] CHIP Constraint Logic Programming Tool, COSYTEC, France.

[Cohen83] P.R. Cohen, E.A. Feigenbaum (1983), *The Handbook of Artificial Intelligence, volume III*, Department of Computer Science, Stanford University, Pitman, pp. 10-27, 475-493, 515-562.

[Currie91] K. Currie and A. Tate (1991), "O-Plan: the open planning architecture", *Artificial Intelligence* **52**, pp. 49-86.

[Dean85] T.L. Dean (1985), *Temporal imagery: an approach to reasoning about time for planning and problem solving*, Technical Report #433, Computer Science Department, Yale University, New Haven, CT.

[Dean86] T.L. Dean (1986), "Handling shared resources in a temporal data base management system", in: *Decision Support Systems* **2**, pp. 135-143.

[Dean87a] T.L. Dean, D.V. McDermott (1987), "Temporal database management", in: *Artificial Intelligence* **32**, pp. 1-22.

[Dean87b] T.L. Dean (1987), "Large-scale temporal data bases for planning in complex domains", in: *Proceedings of the International Joint Conference on AI*, vol. 2.

[Dean88] T.L. Dean, R.J. Firby, D. Miller (1988), "Hierarchical planning involving deadlines, travel time and resources", in: *Computational Intelligence* 4(4), pp. 381-398.

[Dean91] T.L. Dean, M.P. Wellman (1992), *Planning and Control*, Morgan Kaufmann Publishers, San Mateo, CA.

[Dechter91] R. Dechter, I. Meiri, J. Pearl (1991), "Temporal constraint networks", in: *Artificial Intelligence* **49(1-3)**, pp. 61-96.

[Deo74] N. Deo, *Graph Theory with Applications to Engineering and Computer Science*, Prentice Hall, Englewood Cliffs, N.J.

[Doyle79]	J. Doyle (1979), "A truth maintenance system", in: *Artificial Intelligence* **12**, pp. 221-272.
[Entin]	E.E. Entin, D. Serfaty, R.R. Tenney, "Planning with uncertain and conflicting information", in: *Science of Command and Control: part II*, AFCEA International Press, Fairfax, VA.
[Even79]	S. Even (1979), *Graph Algorithms*, Computer Software Engineering Series, Pitman, London.
[Everest92]	G.C. Everest, M.S. Hanna (1992), *Survey of Object-Oriented Database Management Systems*, University of Minnesota, Minneapolis, MN.
[Finger87]	J.J. Finger (1987), *Exploiting constraints in design synthesis*, Phd. Thesis, Stanford University.
[Firby87]	R.J. Firby, D.V. McDermott (1987), "Representing and solving temporal planning problems", in: N. Cercone and G. McCalla (eds.), *The Knowledge Frontier: Essays in the representation of knowledge*, Springer Verlag, New York, pp. 353-413.
[Fox89]	M.S.. Fox (1989), *Reflections on the relationship between Artificial Intelligence and Operations Research*, Carnegie Mellon University, Robotics Institute and Computer Science Department, Pittsburgh, PA.
[Gadsden88]	J.A. Gadsden (1988), "Knowledge-based planning and replanning in Naval Command and Control", in: *Proceedings 4th IEEE Conference on AI Applications*, March 1988, San Diego, CA.
[Georgeff87]	M.P. Georgeff, (1987), "Planning", in: J. Allen, J. Hendler, A. Tate (eds.), *Readings in Planning* , Morgan Kaufmann Publishers, San Mateo, CA, pp. 5-25.
[Gudes91]	E. Gudes, T. Kuflik, A. Meisels (1991), "Limited resource scheduling by generalized rule-based systems", in: *Knowledge Based Systems* **4**, pp. 215-224.
[Hammond90]	K.J. Hammond (1990), "Case-based planning: A framework for planning from experience", in: *Cognitive Science* **14**, pp. 385-443.
[Hayes69]	P. Hayes, J. McCarthy (1969), "Some philosophical problems from the standpoint of Artificial Intelligence", in: *Machine Intelligence* **4**, Edinburgh University Press.
[Henderson86]	R. Mohr, T.C. Henderson (1986), "Arc and path consistency revisited", in: *Artificial Intelligence* **28**, pp. 225-233.
[d'Huy92a]	C.W. d'Huy (1992), *A hybrid domain independent knowledge-based planner*, Report FEL-92-S036, TNO Physics and Electronics Laboratory, The Hague.
[d'Huy92b]	C.W. d'Huy (1992), *Planning en Command & Control op vliegbases: een eerste verkenning* (in Dutch), Report FEL-92-A184, TNO Physics and Electronics Laboratory, The Hague.
[Johnson89]	S.E. Johnson, A.L. Levis (eds.) (1989), *Science of Command and Control: part II*, AFCEA International Press, Fairfax, VA.
[Kautz86]	H. Kautz, M. Vilain (1986), "Constraint propagation algorithms for temporal reasoning", in: *Proceedings AAAI-86*.
[Keene91]	A.P. Keene, M.P. Perre (1991), *Data Fusion: temporal reasoning and truth maintenance*, Report FEL-91-B308, TNO Physics and Electronics Laboratory, The Hague.
[Kleer86a]	J. de Kleer (1986), "An Assumption-based Truth Maintenance System", in: *Artificial Intelligence* **28**, pp. 127-162.
[Kleer86b]	J. de Kleer (1986), "Extending the ATMS", in: *Artificial Intelligence* **28**, pp. 163-196.
[Kleer86c]	J. de Kleer (1986), "Problem solving with the ATMS", in: *Artificial Intelligence* **28**, pp. 197-224.

[Kleer89] J. de Kleer (1989), "A comparison of ATMS and CSP techniques", in: *Proceedings 11th International Joint Conference on Artficial Intelligence*, pp. 290-296.

[Kumar88] V. Kumar, Y. Lin (1988), "A data-dependency-based intelligent backtracking scheme for Prolog", in: *The Journal of Logic Programming* **5(2)**, pp. 165-181.

[Kumar92] V. Kumar (1992), "Algorithms for constraint satisfaction problems: a survey", in: *AI Magazine*, pp. 32-44.

[Lehner86] P.E. Lehner (1986), "On the role of Artificial Intelligence in Command and Control", in: *IEEE Transactions on Systems, Man and Cybernetics* **16(6)**, pp. 824-833.

[Levis89] A.H. Levis, S.E. Johnson (eds.) (1989), *Science of Command and Control: Coping with complexity*, AFCEA International Press, Fairfax, VA.

[Loberg86] G. Loberg, G.M. Powell, A. Oferice, J.D. Roberts (1986), "Representing operational planning knowledge", in: *IEEE Transactions on Systems, Man and Cybernetics* **16(6)**, pp. 774-787.

[Mackworth77] A.K. Mackworth (1977), "Consistency in networks of relations", in: *Artificial Intelligence* **8(1)**, pp. 99-118.

[Maiocchie91] R. Maiocchie, B. Pernici (1991), "Temporal data management systems: a comparitive view", in: *IEEE Transactions on Knowledge and Data Engineering* **3(4)**, pp. 504-523.

[McAllister76] D.F. Stanat, D.F. McAllister (1976), *Discrete Mathematics in Computer Science*, Prentice Hall, Englewood Cliffs, NJ.

[McDermott82] D.V. McDermott (1982), "A temporal logic for reasoning about processes and plans", in: *Cognitive Science* **6**, pp. 101-155.

[Nygard92] K.E. Nygard, C. Yang, "Genetic algorithms for the Travelling Salesman problem with time windows", in: *Computer Science and Operations Research; new developments in their interfaces*, Pergamon Press, Headington Hill Hall, Oxford, UK.

[ONTOS] ONTOS Object-oriented Database Management System, Ontologic Inc., USA.

[POSTGRES] *The POSTGRES Reference Manual*, version 4.0, July 1992, Electronics Research Laboratory, University of California, Berkeley, CA.

[Prolog] *Quintus Prolog IV: Reference Pages*, Quintus Corporation, Palo Alto, CA.

[Sacerdoti77] E.D. Sacerdoti (1977), *A structure for plans and behaviour*, American Elsevier Publishing Company, New York.

[SCHEME] SCHEME Constraint Programming Tool, Honeywell Bull, France.

[Shapiro86] E. Shapiro, L. Sterling, *The Art of Prolog*, MIT press, Cambridge, MA.

[Stallman77] R. Stallman, G.J. Sussman (1977), "Forward reasoning and dependency-directed backtracking", in: *Artificial Intelligence* **9(2)**, pp. 135-196.

[Stonebraker90] M. Stonebraker, L.A. Rowe, M. Hirohama (1990), "The implementation of POSTGRES", in: *IEEE Transactions on Knowledge and Data Engineering* **2(1)**.

[Sutherland90] J.W. Sutherland (1990), "Model-base structures to support adaptive planning in Command and Control systems", in: *IEEE Transactions on Systems, Man and Cybernetics* **20(1)**, pp. 18-32.

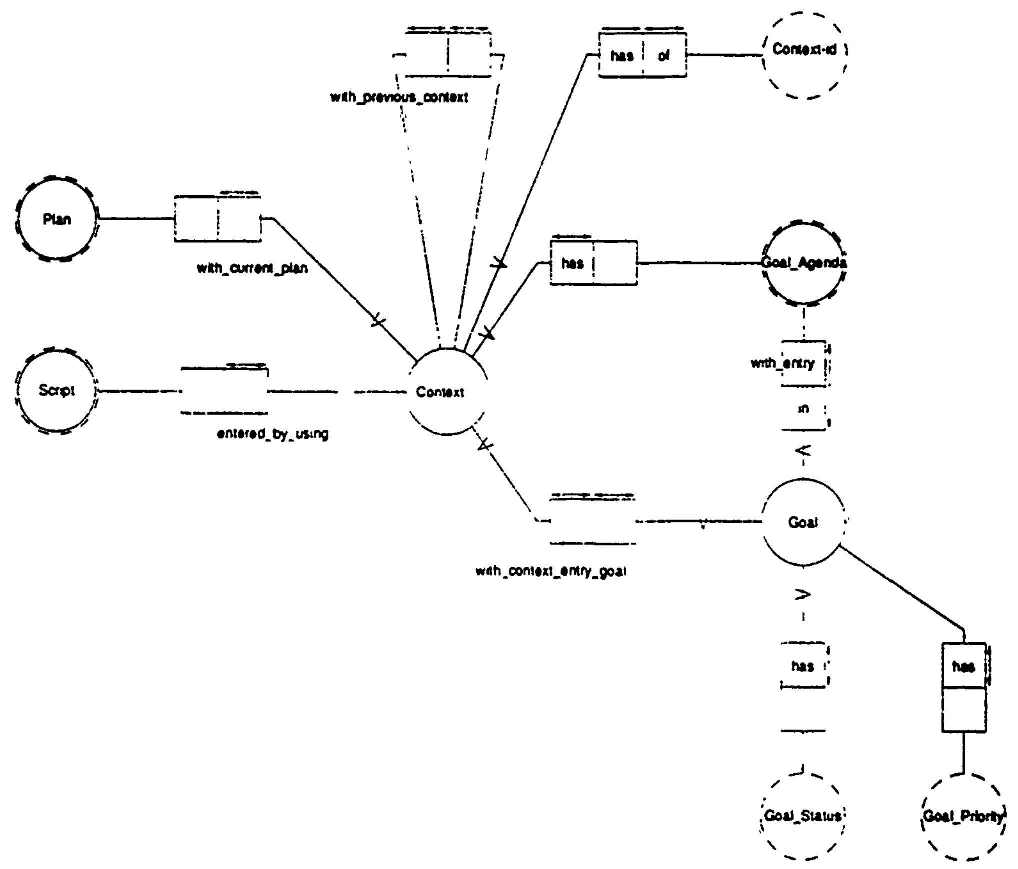
[TeamWork] TeamWork documentation, CADRE Technologies.

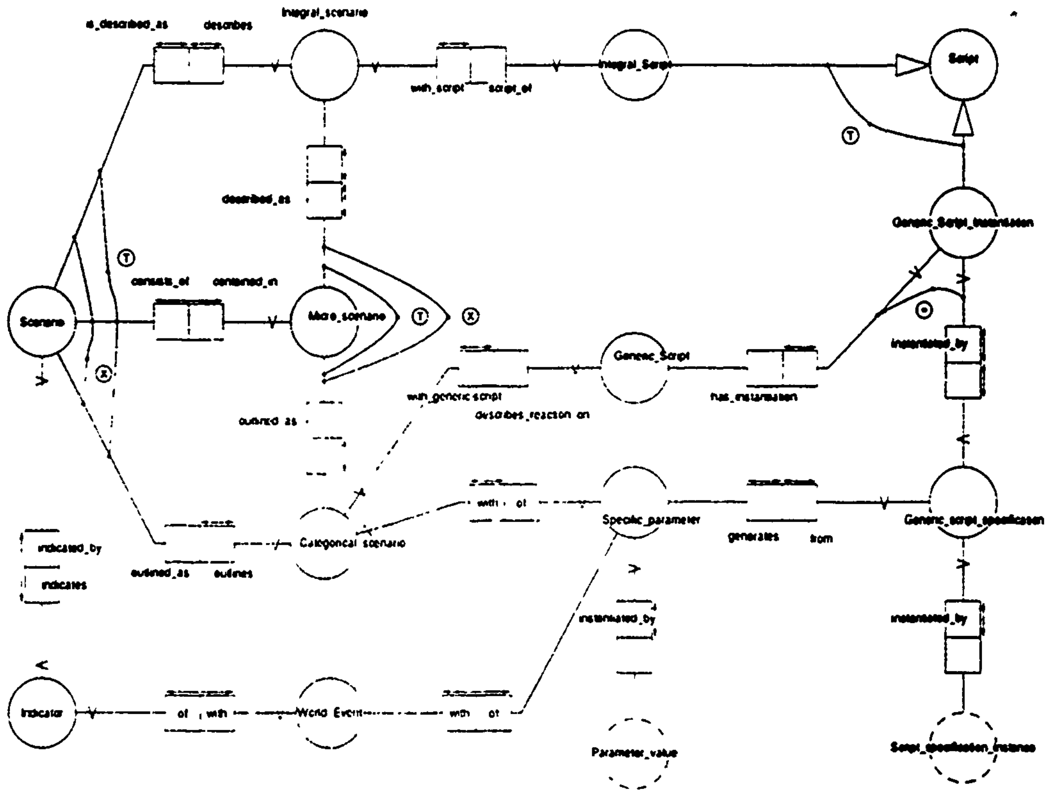
[Vadon91] H. Vadon and A. Tate (eds.) (1991), "Round table on Artificial Intelligence in support of planning and scheduling; working group meeting reports", in: *Proceedings Artificial Intelligence and Knowledge-based Systems for Space*, ESA WPP-025, Noordwijk, The Netherlands.

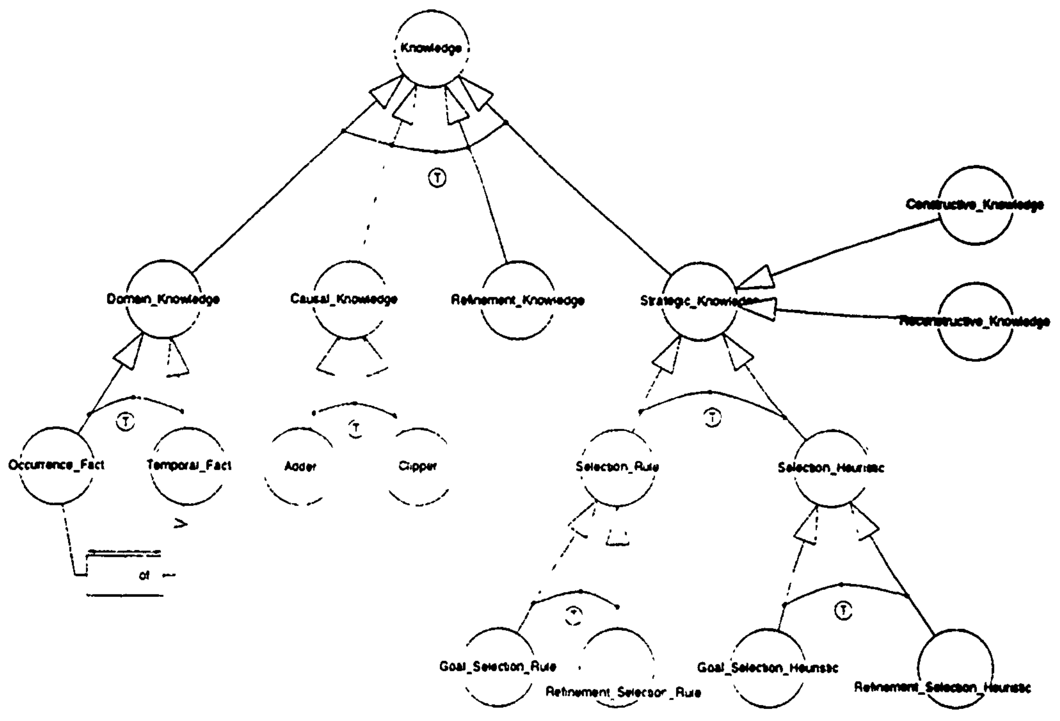
[Witus86] G. Witus (1986), "Decision support for planning and resource allocation in hierarchical organizations", in: *IEEE Transactions on Systems, Man and Cybernetics* **16(6)**, pp. 927-942.

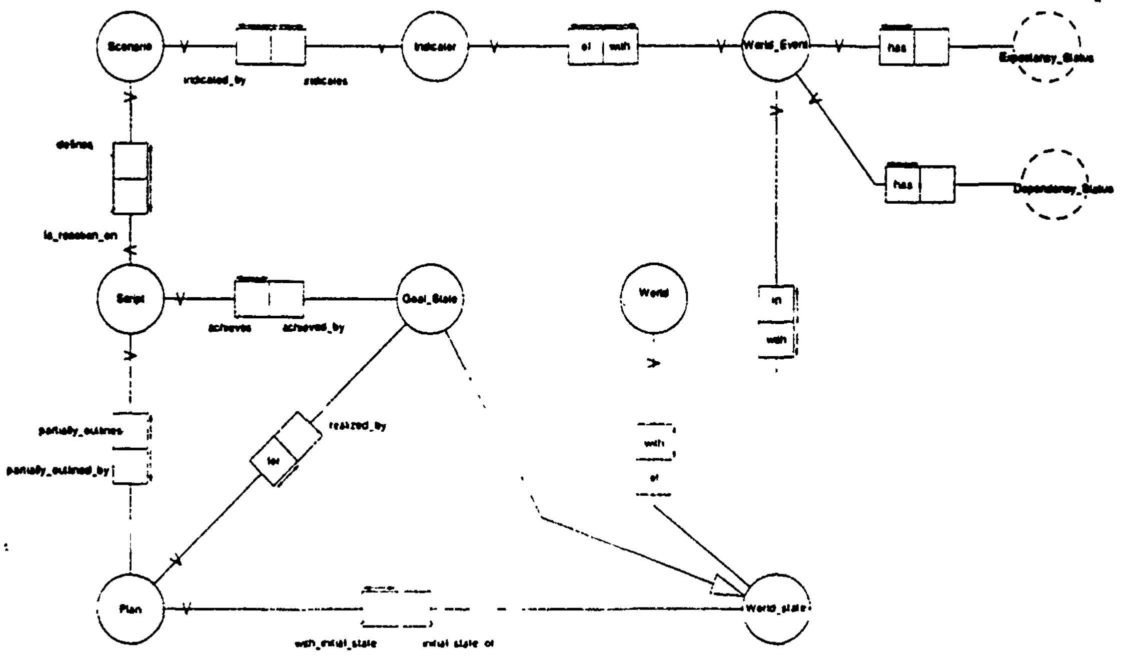
[Wilkins88]	D.E. Wilkins, *Practical Planning: Extending the classical AI planning paradigm*, Morgan Kaufmann Publishers, San Mateo, CA.
[Wintraecken87]	J.J.V.R. Wintraecken (1987), *Informatie-analyse volgens NIAM*, Academic Service, Schoonhoven, The Netherlands (in Dutch).
[Young86]	P.R. Young (1986), "Applications of a theory of automated adversial planning to Command and Control", in: *IEEE Transactions on Systems, Man and Cybernetics* 16(6), pp. 806-812.

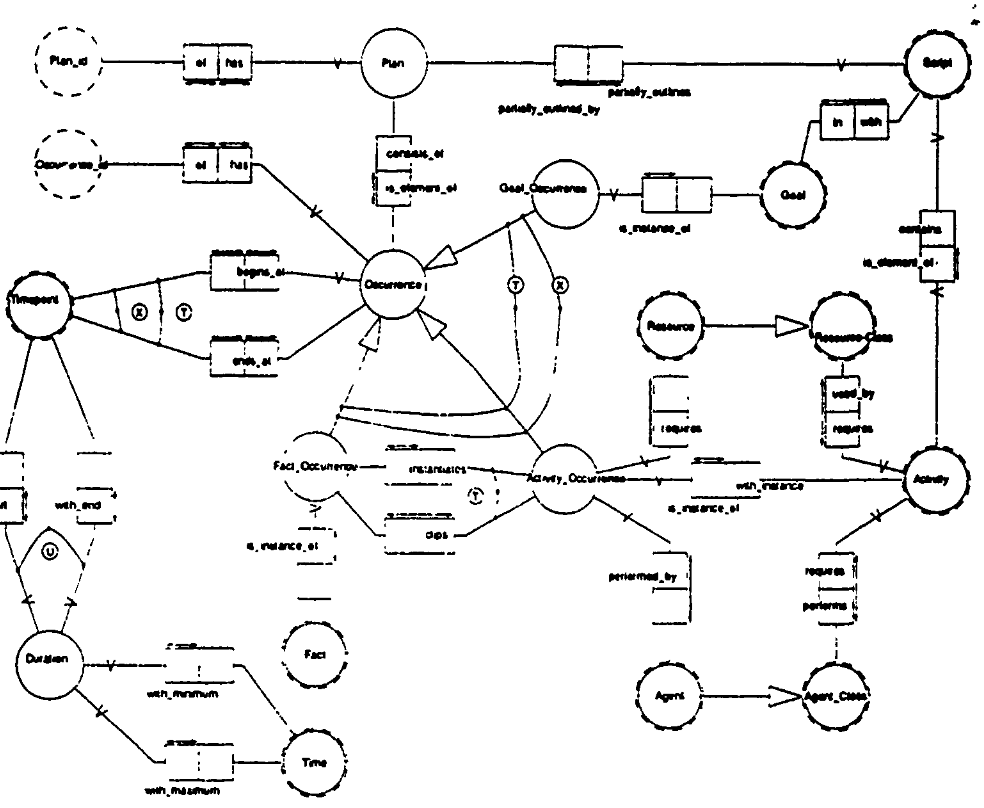
Appendix A: Information structure diagrams

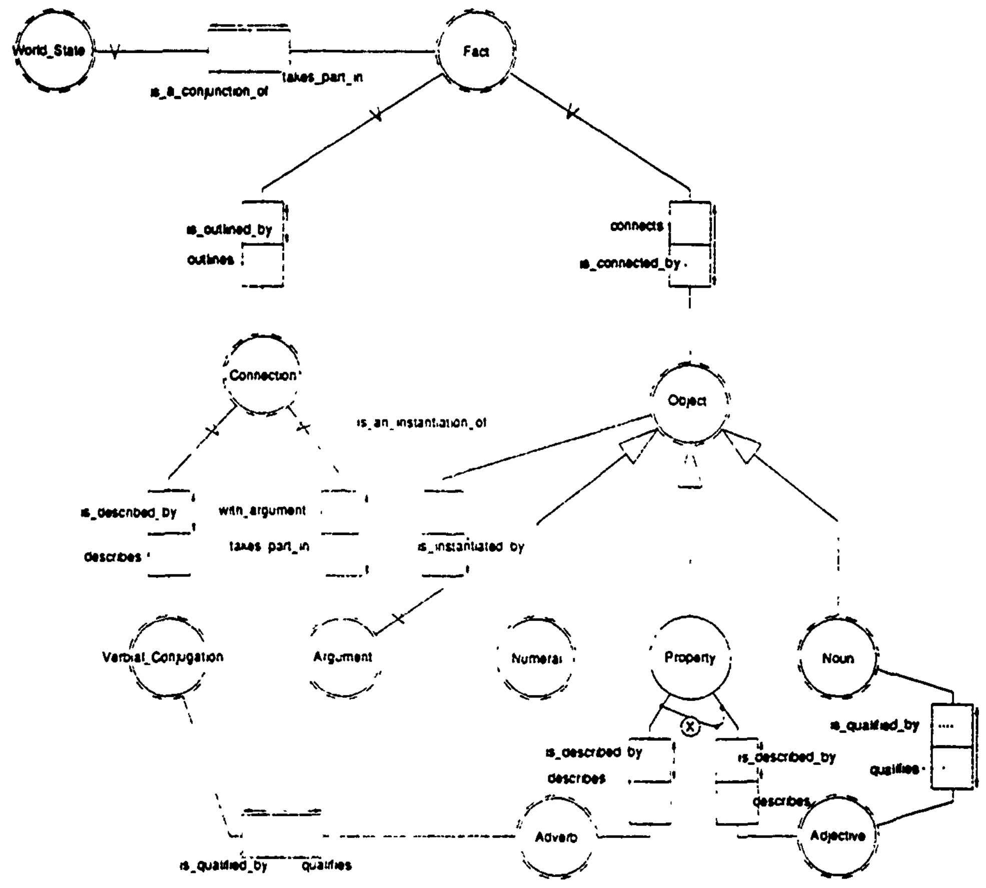

Context-id
has
of
with_previous_context
Plan
with_current_plan
has
Goal_Agenda
with_entry
in
Script
entered_by_using
Context
Goal
with_context_entry_goal
has
has
Goal_Status
Goal_Priority


is_described_as
describes
Integral_scenario
with_script
script_of
Integral_Script
Script
Generic_Script_instantiation
described_as
consists_of
contained_in
Scenario
Micro_scenario
Generic_Script
with_generic script
describes_reaction on
has_instantiation
instantiated_by
outlined as
with
of
Specific_parameter
generates
from
Generic_script_specification
indicated_by
indicates
outlined_as
outlines
Categorical scenario
instantiated_by
Indicator
of
with
World Event
with
of
Parameter_value
Script_specification_instance


Knowledge
T
Constructive_Knowledge
Domain_Knowledge
Causal_Knowledge
Refinement_Knowledge
Strategic_Knowledge
Reconstructive_Knowledge
T
T
T
Occurrence_Fact
Temporal_Fact
Adder
Clipper
Selection_Rule
Selection_Heuristic
of
T
T
Goal_Selection_Rule
Refinement_Selector_Rule
Goal_Selection_Heuristic
Refinement_Selection_Heuristic


Scenario
indicated_by
indicates
Indicator
of
with
World_Event
has
Expectancy_Status
has
Dependency_Status
defines
is_reason_on
Script
achieves
achieved_by
Goal_State
World
in
with
partially_outlines
partially_outlined_by
for
realized_by
with
of
Plan
World_state
with_initial_state
initial state of


Plan_id
of
has
Plan
partially_outlined_by
partially_outlines
Script
Occurrence_id
of
has
consists_of
is_element_of
Goal_Occurrence
is_instance_of
Goal
in
with
contains
is_element_of
Timepoint
begins_at
ends_at
Occurrence
X
T
T
X
Resource
Resource_Class
used_by
requires
requires
Fact_Occurrence
instantiates
clips
T
Activity_Occurrence
with_instance
is_instance_of
Activity
with_start
with_end
U
is_instance_of
performed_by
requires
performs
Duration
with_minimum
with_maximum
Fact
Time
Agent
Agent_Class


World_State
Fact
is_a_conjunction_of
takes_part_in
is_outlined_by
outlines
connects
is_connected_by
Connection
Object
is_an_instantiation_of
is_described_by
describes
with_argument
takes_part_in
is_instantiated_by
Verbal_Conjugation
Argument
Numeral
Property
Noun
is_qualified_by
qualifies
is_described_by
describes
is_described_by
describes
Adverb
Adjective
is_qualified_by
qualifies

ROLES

Each activity requires a agent class.

Each activity requires a resource-class.

Each activity is element of a script.

A activity has a a activity occurrence.

Each activity occurrence requires a resource.

Each activity occurrence is performed by a agent.

Each activity occurrence is instance of at most one activity.

A activity occurrence instantiates a fact occurrence.

A activity occurrence clips a fact occurrence.

A adjective describes a property.

A adjective qualifies a noun.

A adverb describes a property.

A adverb qualifies a verbial conjugation.

A agent performs a activity occurrence.

A agent class performs a activity.

A argument takes part in a connection.

Each argument is instantiated by at most one object.

Each categorical scenario outlines at most one scenario.

Each categorical scenario has a specific parameter.

A categorical scenario is divided into a micro scenario.

Each categorical scenario is counteracted by a generic script.

A connection outlines a fact.

Each connection with argument a argument.

Each connection is described by at most one verbial conjugation.

Each context has at most one context-id.

Each context has at most one goal agenda.

Each context is partial state of at most one plan.

A context is altered by using at most one script.

A context has as previous context at most one andere context.

A context is previous context of at most one andere context.

Each context has as entry goal at most one goal.

A dependency status of a world event.

Each indicator of a world event.

Each indicator indicates a scenario.

Each integral scenario describes at most one scenario.

A integral scenario is divided into at most one micro scenario.

Each integral scenario has at most one integral script.

Each integral script is script of a integral scenario.

Each micro scenario is contained in a scenario.

A micro scenario is described as at most one integral scenario.

A micro scenario is outlined as a categorical scenario.

A noun is qualified by a adjective.

A object is connected by a fact.

A object is an instantiation of a argument.

Each occurrence has at most one occurrence id.

A occurrence is element of at most one plan.

Each occurrence begins at at most one timepoint.

Each occurrence ends at at most one timepoint.

A occurrence fact is ordered by a temporal fact.

A occurrence id of at most one occurrence.

A parameter value of a specific parameter.

Each plan has at most one plan id.

A plan partially outlined by a script.

A plan consists of a occurrence.

A plan is partially outlined by a script.

Each plan for at most one goal state.

Each plan has initial state at most one world state.

Each plan has partial state outlined in a context.

A plan id of at most one plan.

A property is described by at most one adverb.

A property is described by at most one adjective.

A resource is required by a activity occurrence.

Each scenario defines a script.

Each scenario indicated by a indicator.

A scenario is described as at most one integral scenario.

A scenario is outlined as a categorical scenario.

A scenario consists of a micro scenario.

Each scenario is indicated by a indicator.

Each script partially outlines a plan.

Each script partially outlines a plan.

Each script is reaction on a scenario.

Each script achieves at most one goal state.

Each script contains a activity.

Each script alters a context.

A script achieves a goal.

A script specification instance instantiates a generic script specification.

Each specific parameter of a categorical scenario.

A specific parameter generates a generic script specification.

A specific parameter of a world event.

Each specific parameter is instantiated by at most one parameter value.

Each temporal fact orders a occurrence fact.

A time is minimum of a duration.

A time is maximum of a duration.

A timepoint is begin of at most one occurrence.

A timepoint is end of at most one occurrence.

A timepoint is begin of a duration.

A timepoint is end of a duration.

A verbial conjugation describes a connection.

A verbial conjugation is qualified by a adverb.

Each world has a world state.

Each world event in a world state.

Each world event is included in a indicator.

Each world event has a indicator.

A world event has a specific parameter.

Each world event has at most one expectancy status.

Each world event has at most one dependency status.

A world state has a world event.

A world state of at most one world.

A world state is initial state of a plan.

Each world state is a conjunction of a fact.

A context-id of at most one context.

A resource-class is used by a activity.

UNIQUENESS

A duration is uniquely identified by timepoint.

EXCLUSION (set)

activity occurrence, fact occurrence and goal occurrence mutual exclude each other.

EXCLUSION (role)

The roles consists of, is described as and is outlined as of scenario mutual exclude each other.

The roles is begin of and is end of of timepoint mutual exclude each other.

The roles is described as and is outlined as of micro scenario mutual exclude each other.

The roles is described by of property mutual exclude each other.

EQUALITY

The roles _ and is instantiated by of generic script instantiation are equal.

TOTALITY (role)

The roles consists of, is described as and is outlined as of scenario are total.

The roles is begin of and is end of of timepoint are total.

The roles is described as and is outlined as of micro scenario are total.

TOTALITY

The combination of activity occurrence, fact occurrence and goal occurrence is total.

The combination of selection heuristic and selection rule is total.

The combination of occurrence fact and temporal fact is total.

The combination of adder and clipper is total.

The combination of goal selection rule and refinement selection rule is total.

The combination of goal selection heuristic and refinement selection heuristic is total.

SETS

occurrence is a set of activity occurrence, fact occurrence and goal occurrence.

resource-class is a set of resource.

agent class is a set of agent.

world state is a set of goal state.

object is a set of noun, numeral and property.

script is a set of generic script instantiation and integral script.

knowledge is a set of causal knowledge, domain knowledge, refinement knowledge and strategic knowledge.

domain knowledge is a set of occurrence fact and temporal fact.

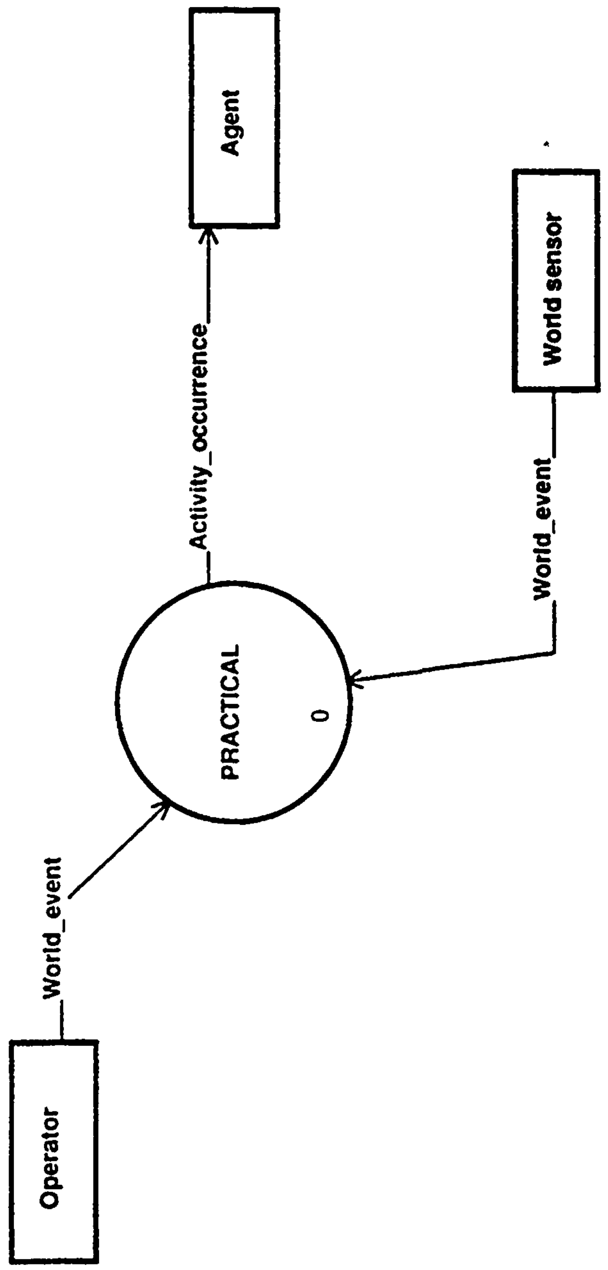
causal knowledge is a set of adder and clipper.

strategic knowledge is a set of constructive knowledge, reconstructive knowledge, selection heuristic and selection rule.

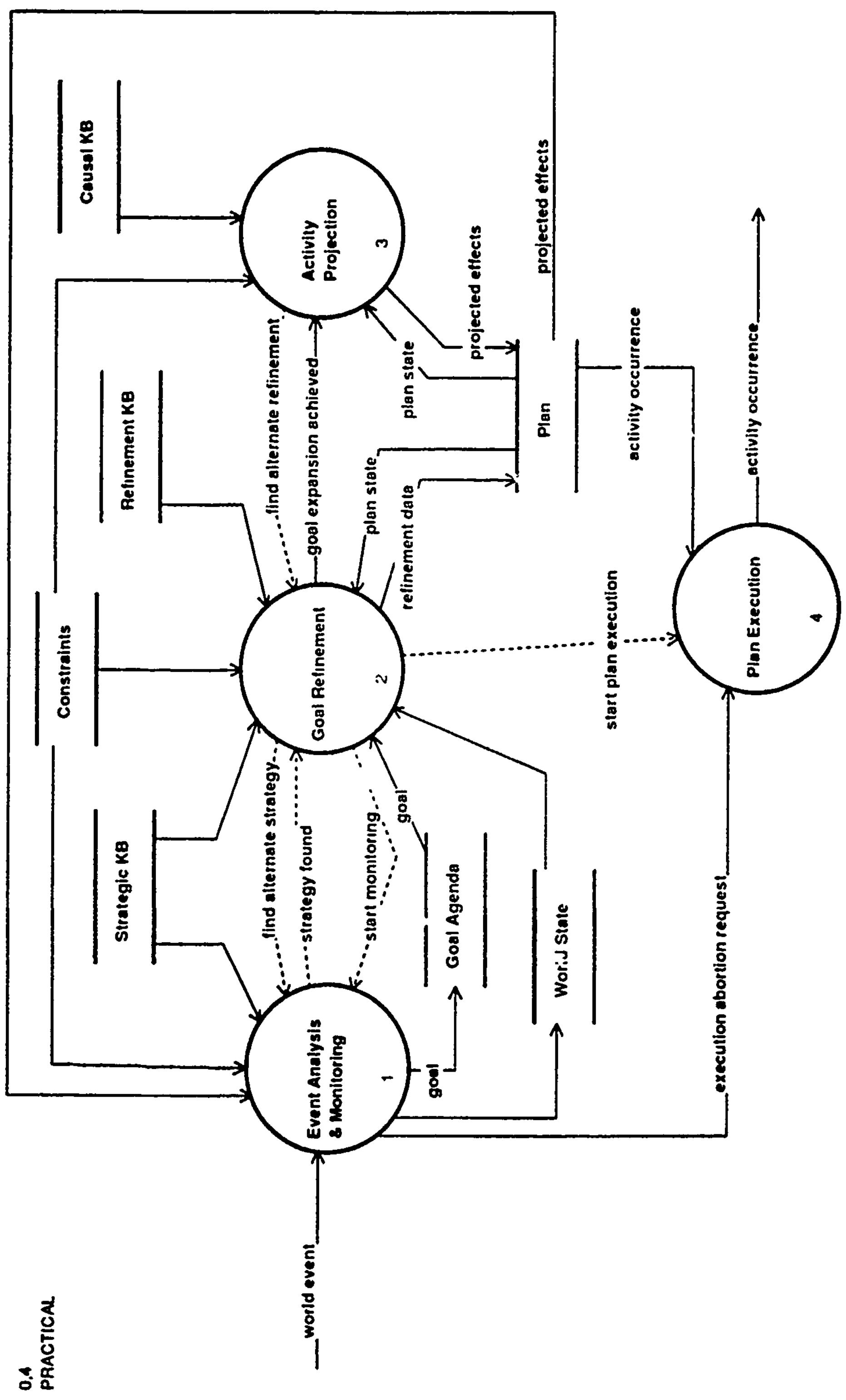
selection heuristic is a set of goal selection heuristic and refinement selection heuristic.

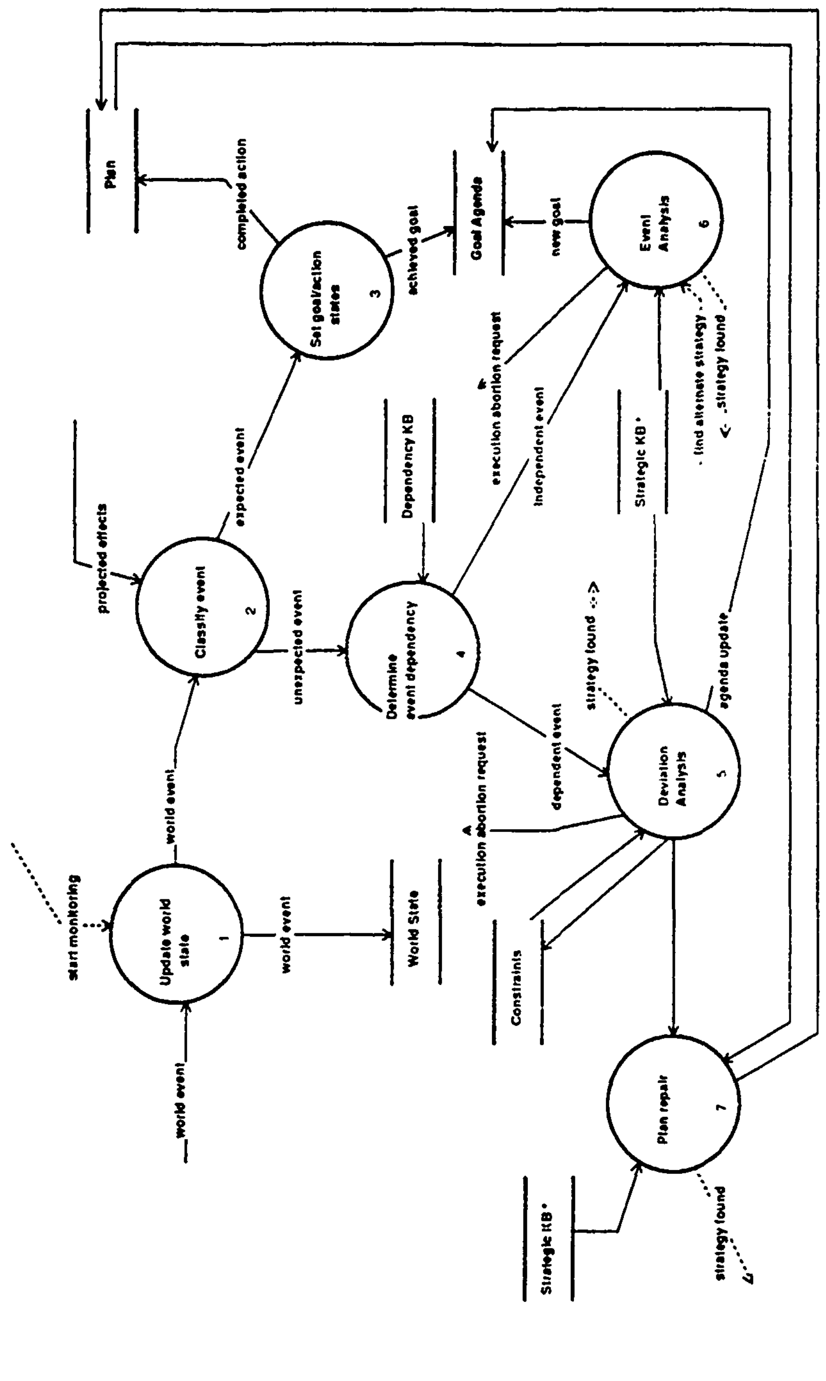
selection rule is a set of goal selection rule and refinement selection rule.

Appendix B: Yourdon data flow diagrams

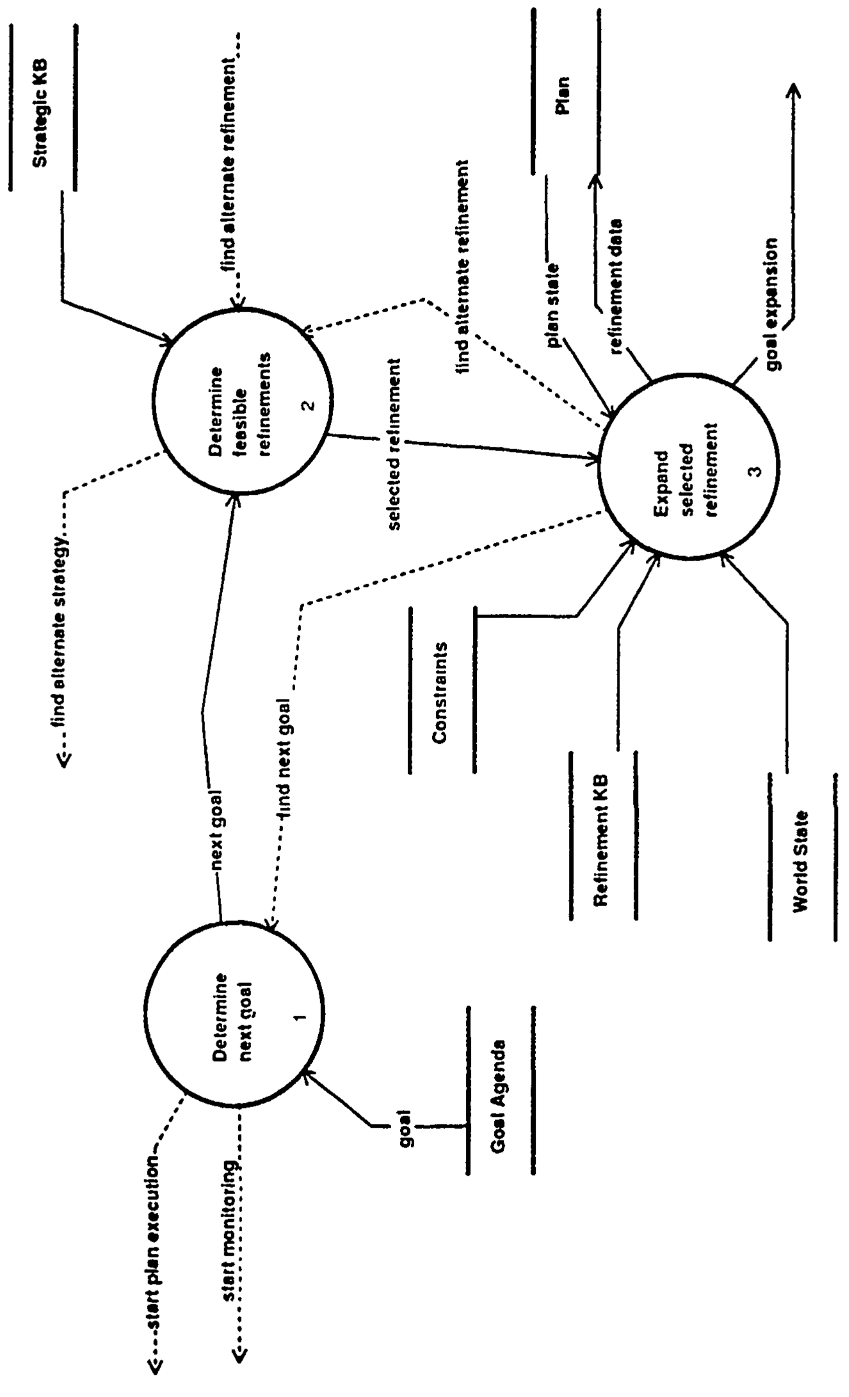


Context-Diagram:2
context diagram

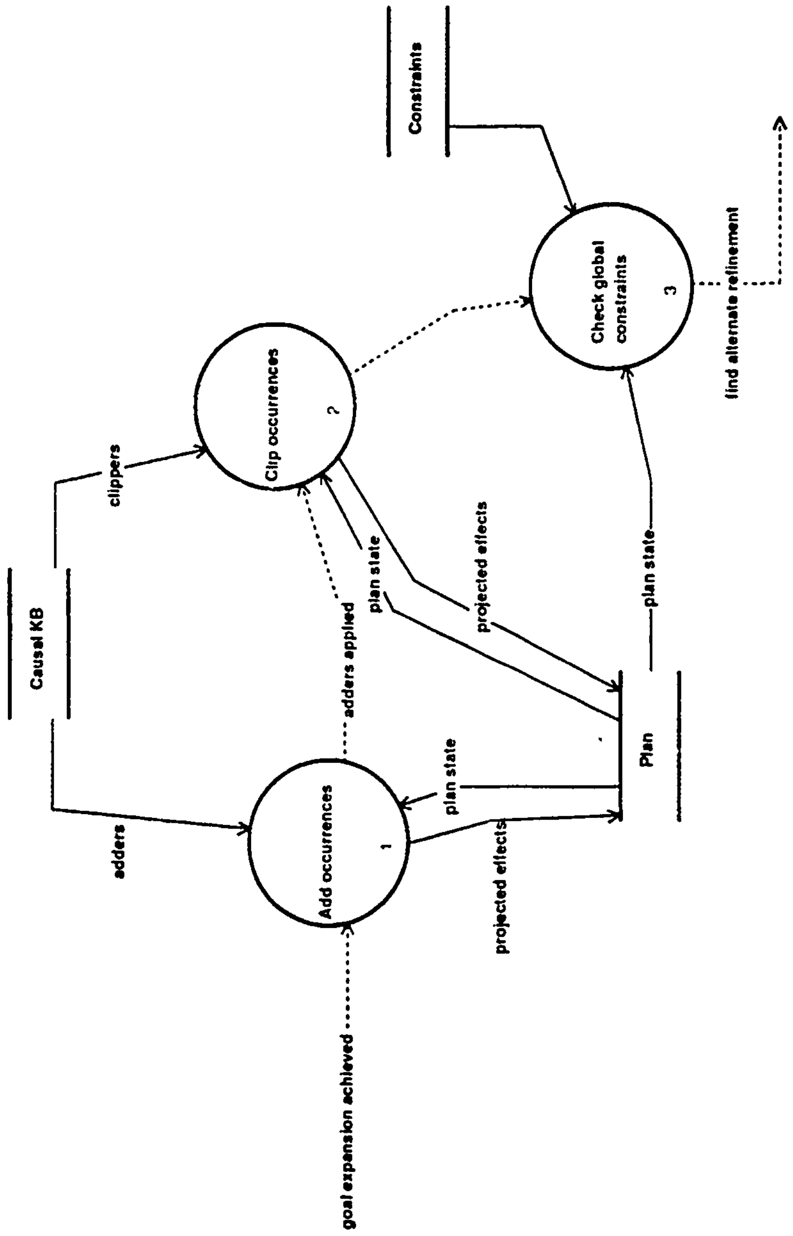

0.4 PRACTICAL
world event
Event Analysis & Monitoring
1
goal
Goal Agenda
World State
execution abortion request
Strategic KB
find alternate strategy
strategy found
start monitoring
goal
Constraints
Goal Refinement
2
start plan execution
Plan Execution
4
activity occurrence
Refinement KB
find alternate refinement
goal expansion achieved
plan state
refinement data
Plan
plan state
projected effects
projected effects
activity occurrence
Causal KB
Activity Projection
3



L,ent Analyss&Monitor ng

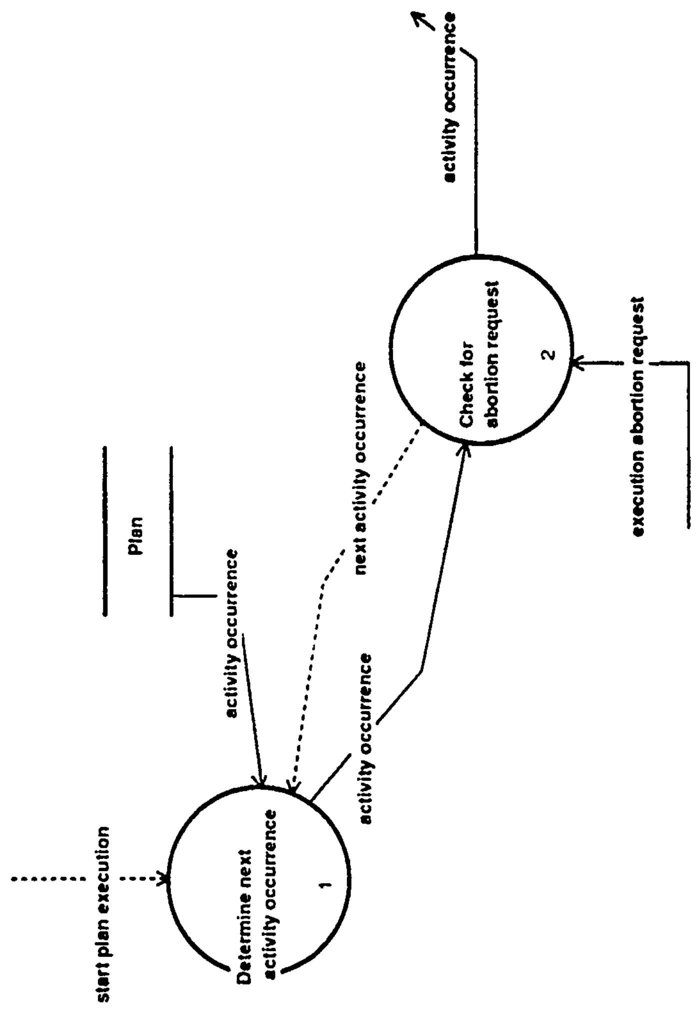


2.2
Goal Refinement



3:1
Activity Projection

4.1
Plan Execution



REPORT DOCUMENTATION PAGE (MOD-NL)

1. DEFENSE REPORT NUMBER (MOD-NL) TD93-0304	2 RECIPIENT'S ACCESSION NUMBER	3. PEF ˉORMING ORGANIZATION REPORT NUMBER FEL-93-B029
4 PROJECT/TASK/WORK UNIT NO 23391	5. CONTRACT NUMBER -	6. REPORT DATE FEBRUARY 1993
7 NUMBER OF PAGES 94 (INCL. 2 APPEND., EXCL. RDP & DIST LIST)	8. NUMBER OF REFERENCES 68	9. TYPE OF REPORT AND DATES COVERED FINAL REPORT

10. TITLE AND SUBTITLE
PRACTICAL: PLANNING AND RESOURCE ALLOCATION IN C2-DOMAINS WITH TIME CRITICAL ALGORITHMS

11. AUTHOR(S)
C.W. D'HUY
A.P. KEENE

12. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
TNO PHYSICS AND ELECTRONICS LABORATORY, P.O. BOX 96864, 2509 JG THE HAGUE
OUDE WAALSDORPERWEG 63, THE HAGUE, THE NETHERLANDS

13. SPONSORING/MONITORING AGENCY NAME(S)
NETHERLANDS MINISTRY OF DEFENCE

14. SUPPLEMENTARY NOTES
THE CLASSIFICATION DESIGNATION ONGERUBRICEERD IS EQUIVALENT TO UNCLASSIFIED.

15. ABSTRACT (MAXIMUM 200 WORDS, 1044 POSITIONS)
PLANNING IS AN EQUALLY IMPORTANT AS COMPLEX PROBLEM IN THE DOMAIN OF COMMAND AND CONTROL. GIVEN AN INCREASING NEED FOR FAST AND FLEXIBLE SOLUTIONS FOR THE ALLOCATION OF RESOURCES, SCHEDULING OF MISSIONS AND PLANNING (LARGE-SCALE) OPERATIONS THE ASSISTANCE OF HUMAN OPERATORS WITH AUTOMATED TOOLS FOR SCHEDULING AND PLANNING BECOMES MANDATORY. REPLANNING IS A CENTRAL ACTIVITY, ALLOWING THE INCORPORATION OF CONTINUOUS CHANGES IN THE ENVIRONMENT INTO THE EVOLVING PLAN. IT IS FURTHERMORE IMPORTANT TO BE ABLE TO ASSESS THE IMPLICATIONS OF OPTIONS IN TERMS OF "WHAT-IF" ANALYSES. THIS REPORT DESCRIBES A FRAMEWORK FOR (RE)PLANNING USING A SCENARIO-DRIVEN APPROACH THAT CENTRES ON METHODS AND TECHNIQUES FOR PLAN GENERATION, PLAN REPAIR AND REPLANNING WITH (PARTIAL) RE-USE OF THE ORIGINAL PLAN. THE BASIS FOR THE FRAMEWORK IS A CONCEPTUAL MODEL OF THE (RE)PLANNING PROCESS AND A KNOWLEDGE PARTITIONING THAT DISTINGUISHES BETWEEN STRATEGIC, TACTICAL, CAUSAL AND WORLD KNOWLEDGE. A TEMPORAL DATABASE MANAGEMENT SYSTEM IS USED TO ALLOW REASONING ABOUT THE PAST AND PRESENT AS WELL AS THE FUTURE. THE RESEARCH HAS CULMINATED IN THE SYSTEM *PRACTICAL*, DEMONSTRATING THE (RE)PLANNING FRAMEWORK, INCORPORATING THE DEMONSTRATORS *CALIGULA* AND *ALLOCATOR* FOR RESOURCE ALLOCATION AND SCHEDULING RESPECTIVELY.

16. DESCRIPTORS	IDENTIFIERS
PLANNING	REPLANNING
REASONING	SCHEDULING
COMMAND & CONTROL	RESOURCE ALLOCATION
ARTIFICIAL INTELLIGENCE	

17a SECURITY CLASSIFICATION (OF REPORT) ONGERUBRICEERD	17b SECURITY CLASSIFICATION (OF PAGE) ONGERUBRICEERD	17c SECURITY CLASSIFICATION (OF ABSTRACT) ONGERUBRICEERD
18. DISTRIBUTION/AVAILABILITY STATEMENT UNLIMITED AVAILABILITY		17d. SECURITY CLASSIFICATION (OF TITLES) ONGERUBRICEERD